MONOGRAFIES DE L'INSTITUT D´INVESTIGACIÓ EN INTEL·LIGÈNCIA ARTIFICIAL

Open, Reusable, and Configurable Multi Agent Systems: A Knowledge Modelling Approach

Mario Gómez Martínez

Consell Superior d´Investigacions Científiques

MONOGRAFIES DE L'INSTITUT D'INVESTIGACIÓ
EN INTEL·LIGÈNCIA ARTIFICIAL
Number 23

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

Monografies de l'Institut d'Investigació en Intel·ligència Artificial

Num. 1 J. Puyol, *MILORD II: A Language for Knowledge–Based Systems*

Num. 2 J. Levy, *The Calculus of Refinements, a Formal Specification Model Based on Inclusions*

Num. 3 Ll. Vila, *On Temporal Representation and Reasoning in Knowledge–Based Systems*

Num. 4 M. Domingo, *An Expert System Architecture for Identification in Biology*

Num. 5 E. Armengol, *A Framework for Integrating Learning and Problem Solving*

Num. 6 J. Ll. Arcos, *The Noos Representation Language*

Num. 7 J. Larrosa, *Algorithms and Heuristics for Total and Partial Constraint Satisfaction*

Num. 8 P. Noriega, *Agent Mediated Auctions: The Fishmarket Metaphor*

Num. 9 F. Manyà, *Proof Procedures for Multiple-Valued Propositional Logics*

Num. 10 W. M. Schorlemmer, *On Specifying and Reasoning with Special Relations*

Num. 11 M. López-Sánchez, *Approaches to Map Generation by means of Collaborative Autonomous Robots*

Num. 12 D. Robertson, *Pragmatics in the Synthesis of Logic Programs*

Num. 13 P. Faratin, *Automated Service Negotiation between Autonomous Computational Agents*

Num. 14 J. A. Rodríguez, *On the Design and Construction of Agent-mediated Electronic Institutions*

Num. 15 T. Alsinet, *Logic Programming with Fuzzy Unification and Imprecise Constants: Possibilistic Semantics and Automated Deduction*

Num. 16 A. Zapico, *On Axiomatic Foundations for Qualitative Decision Theory - A Possibilistic Approach*

Num. 17 A. Valls, *ClusDM: A multiple criteria decision method for heterogeneous data sets*

Num. 18 D. Busquets, *A Multiagent Approach to Qualitative Navigation in Robotics*

Num. 19 M. Esteva, *Electronic Institutions: from specification to development*

Num. 20 J. Sabater, *Trust and Reputation for Agent Societies*

Num. 21 J. Cerquides, *Improved Algorithms for Learning Bayesian Network Classifiers*

Num. 22 M. Villaret, *On Some Variants of Second-Order Unification*

Num. 23 M. Gómez, *Open, Reusable and Configurable Multi-Agent Systems: A Knowledge Modelling Approach*

Num. 24 S. Ramchurn, *Multi-Agent Negotiation Using Trust and Persuasion*

Open, Reusable, and Configurable Multi Agent Systems: A Knowledge Modelling Approach

Mario Gómez Martínez

Foreword by Enric Plaza i Cervera

2005 Consell Superior d'Investigacions Científiques
Institut d'Investigació en Intel·ligència Artificial
Bellaterra, Catalonia, Spain.

Series Editor
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Foreword by
Enric Plaza i Cervera
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Volume Author
Mario Gómez Martínez
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Institut d'Investigació
en Intel·ligència Artificial

Consell Superior
d'Investigacions Científiques

NIPO: 653-05-062-5
ISBN: 84-00-08318-0
Dip. Legal: B-36530-2005

Ordering Information: Text orders should be addressed to the Library of the IIIA, Institut d'Investigació en Intel·ligència Artificial, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain.

A mis padres

Qué ligereza encierra una gota de lluvia
Qué delicado el roce del mundo
Cualquier cosa acontecida en cualquier lugar y tiempo
escrita está en el agua de Babel.

Wislawa Szymborska
(*El Agua*)[1]

En el jeroglífico había un ave, pero no se podía saber si volaba o estaba clavada por un eje de luz en el cielo vacío. Durante centenares de años leí inútilmente la escritura. Hacia el fin de mis días, cuando ya nadie podía creer que nada hubiese sido descifrado, comprendí que el ave a su vez me leía sin saber si en el roto jeroglífico la figura volaba o estaba clavada por un eje de luz en el cielo vacío.

Jose Angel Valente
(*De la luminosa opacidad de los signos*)

Lloras?... Entre los álamos de oro,
lejos, la sombra del amor te aguarda.

Antonio Machado.
(*LXXX*)

[1] "Traducció d'Ana María Moix i Jerzy Wojciech Slawomirski"

Contents

List of Figures

Foreword

The concepts and methodology needed for designing, developing, and implementing real life applications based on multi-agent systems are today still a challenge for researchers in Artificial Intelligence and Computer Science. Industrial-strength multi-agent systems require, among other things, reusability, i.e. the capability of not having to design and implement from scratch a new multi-agent system for every new application domain. Nonetheless, this is the current status of most multi-agent systems developed as of 2004: an ad-hoc multi-agent system is developed for a particular application, a situation that reminds that of expert systems a decade ago.

The Ph. D. monograph of Mario Gómez addresses several dimensions along which multi-agent systems methodology has to be enhanced to achieve reuse. For this purpose, Mario Gómez takes a knowledge modelling stance that comes from the decade long research in knowledge engineering, and adapts its insights to the new challenges risen by multi-agent systems. This approach requires that Mario Gómez is not only aware of the current unsolved issues in multi-agent systems but conversant in the core ideas and developments in knowledge modelling. The resulting contributions are organized in the ORCAS framework, comprising conceptual guidelines to design, methodological commitments for development, and an implemented platform to make concrete these abstract notions and effectively experiment with them.

A main contribution of Mario Gómez ORCAS framework is a clear understanding of how multi-agent systems can be designed and implemented in a way that they are independent of the domain of application, i.e. agents and multi-agent systems can be build in a way their usefulness is not confined to a single application but can be reused for a range of domains. If you wonder how is this possible reading the monograph will be certainly enlightening. Another main contribution is an implemented framework for automatic team formation. The team formation framework puts together at work all the different ideas proposed by Mario Gómez in this framework together with complementary work on multi-agent systems that has been developed in recent years at our Institute. Specifically, the work on electronic institutions is used in the ORCAS framework but at new level not envisioned by the original authors, showing that further developments towards "dynamic institutions" is an addressable challenge.

The ORCAS framework was developed at the Institut d'Investigació en

Intel·ligència Artificial inside the European Commission project *IBROW: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web* and the implemented platform for ORCAS was awarded the *3rd Prize in Agent Infrastructure* in the *Agentcities Agent Technology Competition 2003*.

Bellaterra, July 2004

Enric Plaza i Cervera
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Abstract

Although Multi Agent Systems are supposed to be *open* systems, most of the initial research has focused on *closed* systems, which are designed by one developer team for one homogeneous environment, and one single domain.

This thesis aims to advance some steps towards the realization of the open Multi Agent Systems vision. Our work has been materialized into a framework for developing Multi Agent Systems that maximizes the reuse of agent capabilities across multiple application domains, and supports the automatic, on-demand configuration of agent teams according to stated problem requirements.

On the one hand, this work explores the feasibility of the Problem Solving Methods approach to describe agent capabilities in a way that maximizes their reuse. However, since Problem Solving Methods are not designed for agents, we have had to adapt them to deal with agent specific concepts concerning the agent communication languages and interaction protocols.

One the other hand, this thesis proposes a new model of the Cooperative Problem Solving process that introduces a Knowledge Configuration stage previous to the Team Formation stage. The Knowledge Configuration process performs a bottom-up design of a team in terms of the tasks to be solved, the capabilities required, and the domain knowledge available.

The statements made herein are endorsed by the implementation of an agent infrastructure that has been tested in practice. This infrastructure has been developed according to the electronic institutions formalism to specifying open agent societies. This infrastructure provides a social mediation layer for both requesters and providers of capabilities, without imposing neither an agent architecture, nor an attitudinal theory of cooperation.

The contributions of our work are presented as a multilayered framework, going from the more abstract aspects, to the more concrete, implementation dependent aspects, concluding with the implementation of the agent infrastructure and a particular application example for cooperative information agents.

Agradecimientos

La realización de esta tesis ha sido posible gracias al soporte brindado por el Consejo Superior de Investigaciones Científicas. La mayor parte del trabajo se ha desarrollado bajo la cobertura del proyecto europeo IBROW (IST-1999-190005), y parcialmente bajo el proyecto español SMASH (TIC96-10ajo 38-C04).

Sin el apoyo y el estímulo constante de Enric, mi director de tesis, ésta no se hubiera hecho realidad. Sus recomendaciones y sugerencias transcendieron la labor académica formal y contribuyeron a mi formación integral como investigador en aspectos tales como el respeto frente al trabajo de los otros y la asertividad a la hora de presentar las contribuciones propias.

Esta tesis se ha nutrido de todo eso y mucho más; agradezco a toda la gente del IIIA su compañerismo, su buena disposición a ayudar, y su impagable buen humor, tanto al personal investigador como al personal administrativo y de servicios. Estoy convencido de que la personalidad de las personas es inseparable del ambiente, y viceversa; en ese sentido puedo decir con total sinceridad que el IIIA es un lugar que puede sacar lo mejor de cada uno.

Especial mención merecen los que han colaborado más de cerca en este parto elefantino: Josep Lluís, por su inestimable ayuda con los aspectos más técnicos, sin su implementación de NOOS y su estupenda plataforma para desarrollo de agentes basada en instituciones electrónicas, esta tesis no sería lo que es; a Marc y a todo el equipo de instituciones electrónicas, sin duda otra de las columnas vertebrales de este trabajo; a Santi, autor de la herramienta de visualización Agent World; y a Chema, compañero de casi todo durante estos últimos años, y autor de buena parte del código utilizado en WIM.

Este es el momento de agradecer las historias de pegasos de mi padre, y el regazo de mi madre, más de 11680 días madre, a ellos va dedicada esta tesis. Llegada también la hora de agradecer a mis hermanos su incondicional amistad, es una gran suerte tenerlos como hermanos.

Y por último, infinitas gracias a la persona que me acompañó en este tramo de mi vida, convirtiendo el temido calvario de la tesis en un luminoso paseo.

Chapter 1

Introduction

The main goal of this thesis is to provide a framework for open Multi-Agent Systems that maximizes the reuse of agent capabilities through multiple application domains, and supports the automatic, on-demand configuration of agent teams according to stated problem requirements.

We have devoted considerable effort to the applicability of our proposals, which resulted in the implementation of an infrastructure to develop Multi Agent Systems according to the principles and requirements stated by our framework.

During the rest of this Chapter the main goal of this thesis is analyzed and boiled down to the several issues and problems it encompasses. First, we situate our work in the field of Multi-Agent Systems, focusing on the open problems and challenges that motivated us; second, the main contributions of this thesis are summarized; and third, the structure of the thesis is presented as a guide for readers.

1.1 Motivation and context

Distributed Artificial Intelligence has historically been divided in two main areas: Distributed Problem Solving (DPS) and Multi-Agent Systems (MAS) [Bond and Gasser, 1988a]. In the DPS approach problems are divided and distributed among a number of nodes that cooperate in solving the different parts of the problem; but the overall problem solving strategy is an integral part of the system. In contrast, MAS research is concerned with the behavior of a collection of possibly pre-existing autonomous agents aiming at solving a given problem [Jennings et al., 1998]. MAS have been defined as loosely coupled networks of problem-solving entities working together to find answers to problems that are beyond the individual capabilities or knowledge of the isolated entities [Durfee and Lesser, 1989]. The MAS approach advocates decomposing problems in terms of autonomous agents that can engage in flexible, high level interactions, and this way of decomposing a problem aids the process of engineering complex systems [Jennings, 2000]. Some characteristics of MAS are the following:

- each agent has incomplete information or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized; and
- computation is asynchronous.

Some reasons for the increasing interest in MAS research include: the ability to provide robustness and efficiency, the ability to allow inter-operation of existing legacy systems, and the ability to solve problems in which data, expertise, or control is distributed. Agents are defined as sophisticated computer programs that act autonomously on behalf of their users, across open and distributed environments, to solve a growing number of complex problems.

Considering the former definitions, MAS are supposed to be open systems in that agents can enter / leave at any time. Nonetheless, most of the initial work devoted to MAS research has focused on *closed* systems [Klein, 2000], typically designed by one team for one homogeneous environment, with participating agents sharing common high-level goals in a single domain. The communications languages and interaction protocols are typically in-house protocols, and are defined by the design team prior to any agent interactions. Systems are scalable under controlled conditions and design approaches tend to be ad hoc, inspired by the agent paradigm rather than using any specific methodologies.

It is often suggested the need for real open systems that were capable of dynamically adapting themselves to changing environments. Some examples are electronic markets, communities and distributed search engines. All in all, in open MAS the participants (both human and software agents) are unknown beforehand, can change over time and can be developed by different parties. Open systems are opposite to closed or proprietary systems, i.e. open systems can be supplied by hardware components from multiple vendors, and whose software can be operated from different platforms.

According to the predictions of the European Network of Excellence for Agent Based Computing, fully open MAS spanning multiple application domains and involving heterogeneous participants will not be achieved in a foreseeable future, and not before year 2009 [Luck et al., 2003]. This cautious prediction obeys to some challenges yet to be undertaken, including the following:

- provide effective agreed standards to allow open agent systems;
- provide semantic infrastructure for open agent communities;
- develop reasoning capabilities for agents in open environments;
- develop agent ability to understand user requirements;
- develop agent ability to adapt to changes in the environment;
- ensure agent confidence an trust in agents

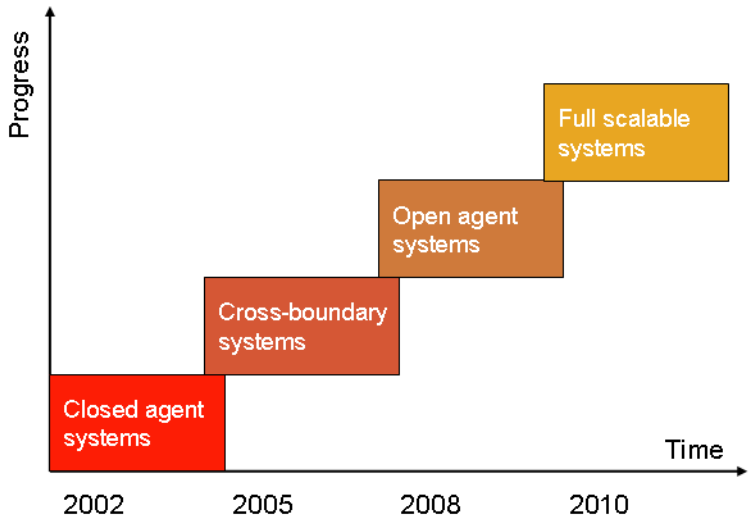


Figure 1.1: Roadmap timeline for agent technologies

Figure 1.1 (adapted from [Luck et al., 2003]) shows a roadmap timeline suggesting how agent technology will progress over time if R & D is aimed at the main challenges identified.

Although we are not going to solve all these problems entirely, we hope to provide tentative solutions to some of them and to bring about some insights that could drive future work on these issues. We are not going to exhaustively describe these problems here, since they are described in Chapter 2; we are rather going to sketch them so as to let the reader become acquainted with the motivations for this work.

Nowadays, MAS are increasingly being designed to cross corporate boundaries, so that the participating agents have fewer goals in common, although their interactions are still concerning a common domain. The languages and protocols used in these systems are being agreed and standardized; however, despite this raising diversity, all participating agents are designed by the same team designing the system and share common domain knowledge.

In order to overcome the limitations of current agent infrastructures for open MAS, researchers must tackle with several problems:

- the connection problem, or how to put providers and requesters (services and customers) in contact;

- the interoperability problem, or how to achieve a meaningful interaction among heterogeneous agents at the syntactic, semantic and pragmatic levels;

- the coalition problem, or how to form and coordinate agent teams to solve problems in a cooperative way;
- the reuse problem, or how to use the same agent capabilities across several application domains;
- the accountability problem, or how to predict or explain the behavior of MAS according to the requirements of the problem.

The MAS community builds intelligent agents capable of reasoning about how to cooperate to solve complex problems. This area uses knowledge intensively: for adding meaning (using ontologies), enabling service discovery and composition (using annotations and reasoning for matchmaking), and coordinating processes (using negotiation strategies). In closed environments knowledge is usually homogeneous and static. In open environments such as the Internet, knowledge is pervasive, distributed, heterogeneous, and dynamic in nature.

Nowadays the Web is shifting the nature of software development to a distributed *plug-and-play* process. This change requires a new way of managing and integrating software based on a software integration architectural pattern called *middleware*. Middleware is connectivity software; it consists of enabling services that allow multiple processes running on one or more machines to interact across a network. It follows that a middleware layer is required to provide a common set of programming interfaces that developers can use to create distributed systems.

Intelligent middleware aims to achieve the highest degree of interoperability, where systems can identify and react to the semantics of data. For this reason, many research communities are focusing their attention to semantic interoperability, for example: MAS, Semantic Web Services, Cooperative Information Systems and Component Based Software Development.

In open MAS the middleware layer is usually provided by *middle agents* [Decker et al., 1997b] that mediate between requesters and providers of capabilities, e.g. matchmakers [Decker et al., 1996], facilitators [Erickson, 1996a, Genesereth and Ketchpel, 1997] and brokers[Nodine et al., 1999]). Typically, the function of a middle agent is to pair requesters with providers that are suitable for them, and this process is called *matchmaking*. To enable matchmaking, both providers and requesters share a common language to describe the requests (tasks or goals) and the advertisements (capabilities or services) in order to compare them. This language is called an Agent Capability Description Language (ACDL).

Matchmaking is the process of verifying whether a capability specification matches the specification of a request (e.g. a task to be solved): two specifications match if their specifications verify some *matching* relation, where the matching relation is defined according to some criteria (e.g. a capability being able to solve a task). Semantic matchtmaking, which is based on the use of shared ontologies to annotate agent capabilities [Guarino, 1997a], improves the matchmaking process and facilitates interoperation.

Semantic matchmaking allows to verify whether a capability can solve a new type of problem (a task), but the reuse of existing capabilities over new

application domains is difficult because capabilities are usually associated to a specific application domain.

The notion of an *Agent Capability Description Language* (ACDL) has been introduced recently [Sycara et al., 1999a] as a key element to enable MAS interoperation in open environments. An ACDL is a shared language that allows heterogeneous agents to coordinate effectively across distributed networks. Sometimes, capabilities are referred as "services" and, consequently, an ACDL can alternatively be called an Agent Service Description Language (ASDL).

In the literature, an ACDL is defined as a language to describe both agent advertisements and requests, and is primarily used by middle agents (e.g. brokers and matchmakers) to pair service-requests with service-providing agents that meet the requirements of the request [Sycara et al., 1999b, Sycara et al., 1999a].

Some desirable features for such a language are *expressiveness*, *efficiency* and *ease of use*:

- *Expressiveness*: the language should be expressive enough to represent not only data and knowledge, but also the meaning of a capability. Agent capabilities should be described at an abstract rather than implementation dependent level.
- *Efficiency*: inferences on descriptions written in this language should be supported. Automatic reasoning and comparison on the descriptions should be both feasible and efficient.
- *Ease of use*: descriptions should not only be easy to read and understand, but also easy to write. The language should support the use of ontologies for annotate agent capabilities with shared semantic information.

However, in addition to capability discovery, an ACDL should bring support to other activities involved in MAS interoperation. On the one hand, once a capability is discovered, it should be enacted automatically; agents should be able to interpret the description of a capability to understand what input is necessary to execute a capability, what information will be returned, and which are the effects or postconditions that will hold after applying the capability. In addition, an agent must know the communication protocol, the communication language and the data format required by the provider of the capability in order to successfully communicate with it.

On the other hand, in order to achieve more complex tasks, capabilities may be combined or aggregated to achieve complex goals that existing capabilities cannot achieve in isolation. This process may require a combination of matchmaking, capability selection among alternative candidates, and verification of whether the aggregated functionality satisfies the specification of a high-level goal.

Our approach to these activities is tightly related with the idea of reuse: how to reuse a capability for different tasks, across several application domains, and in cooperation with other capabilities provided by different, probably heterogeneous agents. The idea of reuse is being addressed by the Software Engineering and the Knowledge Engineering communities.

The reuse of complete software developments and the process used to create them has the potential to significantly ease the process of software engineering by providing a source of verified software artifacts [Wegner, 1984]. It is suggested than reuse of software artifacts can be achieved through the utilization of software libraries [Atkinson, 1997]. Essentially a software library is a repository of information which can be used to construct software systems. The main goal of software libraries reuse is to enable previous development experiences to guide subsequent software development. To this end, MAS designers must be provided with libraries of:

- generic organisation models (e.g., hierarchical organisations, flat organisations);
- generic agent models (e.g., purely reactive agent models, deliberative BDI models);
- generic task models (e.g., diagnostic tasks, information filtering tasks, transactions);
- communication languages and patterns for agent societies;
- ontology patterns for agent requirements, agent models and organisation models;
- interaction protocol patterns between agents with special roles;
- reusable organisation structures; and
- reusable knowledge bases.

From the compositional approach, building a software system is essentially a design problem [Biggerstaff and Perlis, 1989]. The Component-Based Software development (CBSD) approach focuses on building large software systems by integrating previously-existing software components. By enhancing the flexibility and maintainability of systems, the ultimate goal is to reduce software development costs, assemble systems rapidly, and reduce the maintenance burden associated with the support and upgrade of large systems [Brown and Wallnau, 1996].

Constructing an application involves the use of prefabricated pieces, perhaps developed at different times, by different people and possibly with different purposes, therefore integrability of heterogeneous components is a key when considering whether to acquire, reuse, or build new components. Reusable software components can be deployed independently and are subject to composition by third parties [Szyperski, 1996]. There is, however, a major problem with software composition, the so called *Bottom Up Design Problem* [Mili et al., 1995], defined as:

> given a set of requirements, find a set of components within a software library whose combined behavior satisfies the requirements.

The fundamental difficulty when considering this problem is how to decompose the requirements in such a way as to yield component specifications. A reverse approach is to search the space of all possible component compositions until one satisfying the requirements is found [Hall, 1993, Zhang, 2000]. Thus, composition of components can be regarded as composition of their specifications [Butler and Duke, 1998].

Concerning Knowledge Engineering, we are interested in Knowledge Modelling Frameworks that has proposed several methodologies, architectures and languages for analyzing, describing and developing knowledge systems [Steels, 1990, McDermott, 1988, Schreiber et al., 1994a, Fensel et al., 1999]. The goal of a Knowledge Modelling Framework (KMF) is to provide a conceptual model of a system which describes the required knowledge and inferences at an implementation independent way. This approach is intended to support the engineer in the knowledge acquisition phase [Van de Velde, 1993] and to facilitate reuse [Fensel, 1997a].

However, KMFs and reusable software libraries have rarely been applied in the field of MAS to deal with the reuse and interoperation problems arising in open environments. This thesis explores the utility of a KMF to support the automated design and coordination of agent teams according to stated problem requirements; in other words, we translate the Bottom Up Design Problem problem to the MAS field: given a set of requirements, find a set of agent capabilities whose combined competence and knowledge satisfy the requirements.

1.2 Contributions

The main outcome of our efforts to overcome the problems concerning interoperability and reuse in open MAS is a multi-layered framework for MAS development and deployment that integrates Knowledge Modelling and Cooperative Multi-Agent Systems together. This framework is called ORCAS, which stands for Open, Reusable and Configurable multi-Agent Systems.

The ORCAS framework explores the use of a KMF for describing and composing agent capabilities with the aim of maximizing capability reuse and supporting the automatic, on-demand configuration of agent teams according to stated problem requirements. The ORCAS KMF is being used as an ACDL supporting semantic matchmaking and allowing capability descriptions in a domain independent manner, in order to maximize capability reuse.

The Knowledge Modelling Framework of ORCAS has been complemented with an Operational Framework, which describes a mapping from concepts in the Knowledge-Modelling Framework to concepts from Multi-Agent Systems and Cooperative Problem Solving. Specifically, the Operational Framework describes how a composition of capabilities represented at the knowledge-level can be operationalized by a customized team of problem solving agents. In order to do that, the Operational Framework extends the KMF to describe also the communication and the coordination mechanisms required by agents to cooperate. Our approach to describe such aspects of a capability is based on the macro-level

(societal) aspects of agent societies, which is focused on the communication and the observable behavior of agents, rather than adopting a micro-level (internal) view on individual agents. The reason to focus on the macro-level is to avoid imposing a specific agent architecture, thus facilitating the design and development of agents to third parties, a basic requirement of open MAS.

The ORCAS Operational Framework proposes a new model of the Cooperative Problem Solving process that is based on a knowledge-level [Newell, 1982] description of agent capabilities, using the ORCAS KMF. This model includes a Knowledge Configuration process that takes a specification of problem requirements as input and searches a composition of capabilities and knowledge satisfying those requirements. The result of the Knowledge Configuration process is a *task-configuration*, a knowledge-level design of an abstract agent team, in terms of the tasks to be solved, the capabilities to be applied, and the knowledge to be used by those capabilities.

An agent willing to start a cooperative activity requires an initial plan to know which are the capabilities required in order to select suitable agents for that plan. In larger systems, team selection may involve an exponential number of possible team combinations, and a blow-out in the number of interactions required to select the members of a team. There are two approaches to overcome these problems: one approach, that still relies on some kind of global plan is that of guiding the team formation with problem requirements [Tidhar et al., 1996]; another approach is to use distributed tasks allocation methods to make the team selection computationally tractable [Shehory and Kraus, 1998, Sandholm, 1993]; furthermore, a mixture of both approaches is also feasible [Clement and Durfee, 1999].

In this thesis we adopt the approach based on guiding the team formation process with the problem requirements, but the notion of a initial plan is here replaced by the notion of a task-configuration. A task-configuration reduces the complexity of the team formation process by constraining the composition of the team to a certain design that satisfies the requirements of the problem. In spite of its combinatorial nature, the complexity of the team selection process is mitigated, though partially transferred from the team formation process to the Knowledge Configuration process. Therefore, in order to further reduce the complexity of the Knowledge Configuration and the team formation activities, we propose Case-Based Reasoning to heuristically guide the search process over the space of possible configurations.

Finally, we have implemented and agent infrastructure according to the ORCAS model of the CPS process. This agent infrastructure has being implemented using the *electronic institutions* formalism [Esteva et al., 2001, Esteva et al., 2002b], which is based on a computational metaphor of human institutions from a macro-level point of view.

Human institutions are places where people meet to achieve some goals following specific procedures, e.g. auction houses, parliaments, stock exchange markets, etc. Intuitively, the notion of electronic institutions refers to a sort of virtual place where agents interact according to explicit conventions. The

institution is the responsible for defining the rules of the game, to enforce them and impose the penalties in case of violation.

An electronic institution, or e-Institution, is a "virtual place" designed to support and facilitate certain goals to the human and software agents concurring to that place. Since these goals are achieved by means of the interaction of agents, an e-institution provides the social mediation layer required to achieve a successful interaction: interaction protocols, shared ontologies, communication languages and social behavior rules. The interaction is not only regulated by the institution, furthermore it is mediated by institutional agents that offer an added value to participating agents.

The ORCAS e-Institution brings an added value to both requesters and providers: on the one hand, requesters are freed of finding adequate providers and provides a single interface to the multiple and heterogenous providers; on the other hand, the institution provides an advertisement service to capability providers, provides a mediation service for the team formation process, and facilitates coordination during the teamwork activity, allowing agents to solve complex problems that cannot be achieved by an agent alone.

However, in addition to implement an agent infrastructure using the electronic institutions formalism, we are interested on using the concepts proposed by the e-Institutions approach to describe the communication and the operational description of agent capabilities without imposing neither a specific agent architecture, nor an attitudinal theory of cooperation.

The goal of partitioning the ORCAS framework in layers is to bring developers an extra flexibility in adapting this framework to their own requirements, preferences and needs. We claim that a clear separation of layers will support a flexible utilization and extension of the framework to fit different needs, and to build different infrastructures. Therefore, we divide the ORCAS framework in three complementary frameworks:

1. The *Knowledge Modelling Framework* (KMF) proposes a conceptual and architectural description of problem-solving systems from a knowledge-level view, abstracting the specification of components from implementation details. In addition, a Knowledge Configuration model is presented as the process of finding configurations of components that fulfill stated problem requirements.

2. The *Operational Framework* deals with the link between the characterization of components and its implementation, that in our framework is realized by Multi-Agent Systems. This framework comprehends an extension of the KMF to become a full-fledged Agent Capability Description Language, together with a new model of the Cooperative Problem Solving process based on the KMF.

3. The *Institutional Framework* describes an implemented infrastructure for developing and deploying Multi-Agent Systems configurable on-demand, according to the the two layered —knowledge and operational— configuration framework. This infrastructure is designed and implemented ac-

cording to an institutional model of open agent societies. The result is multi-agent platform that supports flexible, extensible and configurable Multi-Agent Systems.

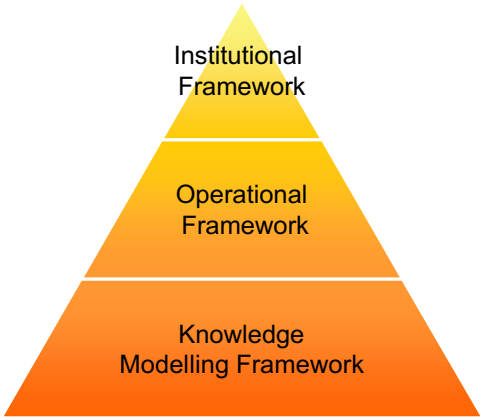


Figure 1.2: The three layers of the ORCAS framework

Figure 1.2 shows the three layers as a pyramid made of three blocks. The block at the bottom corresponds to the more abstract layer, while upper blocks corresponds to increasingly implementation dependent layers. Therefore, developers and system engineers can decide to use only a portion of the framework, starting from the bottom, and modifying or changing the other frameworks according to its preferences and needs.

1.3 Structure

This thesis consist of 7 chapters, including this one, and several appendixes providing technical information. The thesis is organized as follows (Figure 1.3):

Chapter 2 reviews some research relevant to our thesis and discusses some of their contributions that put the basis for our work, together with its limitations and the open issues we are dealing with. Since our work integrates two fields together -knowledge modelling and multi-agent systems-, this chapter have to address very different issues.

Chapter 3 draws the structure of ORCAS framework to give the reader an overall view of it, and remarks the outstanding elements of each layer so as to disclose the logic underpinning that structure.

Chapter 4 proposes a knowledge modelling framework for Multi-Agent Systems. This framework describes a conceptual and architectural characterization of problem-solving systems from a knowledge-level perspective,

abstracting the specification from any implementation details. Moreover, this chapter describes a Knowledge Configuration process that is able to find a configuration of components (tasks, capabilities and domain-models) fulfilling stated problem requirements.

Chapter 5 describes a framework to operationalize a knowledge-level configuration by forming and instructing a team of agents with the required capabilities and domain knowledge. This chapter describes also a model of teamwork based on the social view on agent cooperation.

Chapter 6 introduces the institutional framework, an implemented infrastructure for system development that is based on the two layered approach to multi-agent configuration together with an institutional approach to open agent societies, in support of flexible, customizable and extensible Cooperative Multi-Agent Systems.

Chapter 7 shows an implemented application as a case study of the ORCAS framework, the Web Information Mediator (WIM). WIM is an application to look for medical bibliography in Internet that relies on a library of tasks and agent capabilities for information search and aggregation, linked to a medical application domain.

Chapter 8 presents some conclusions and draws up those open issues that are believed to deceive future work.

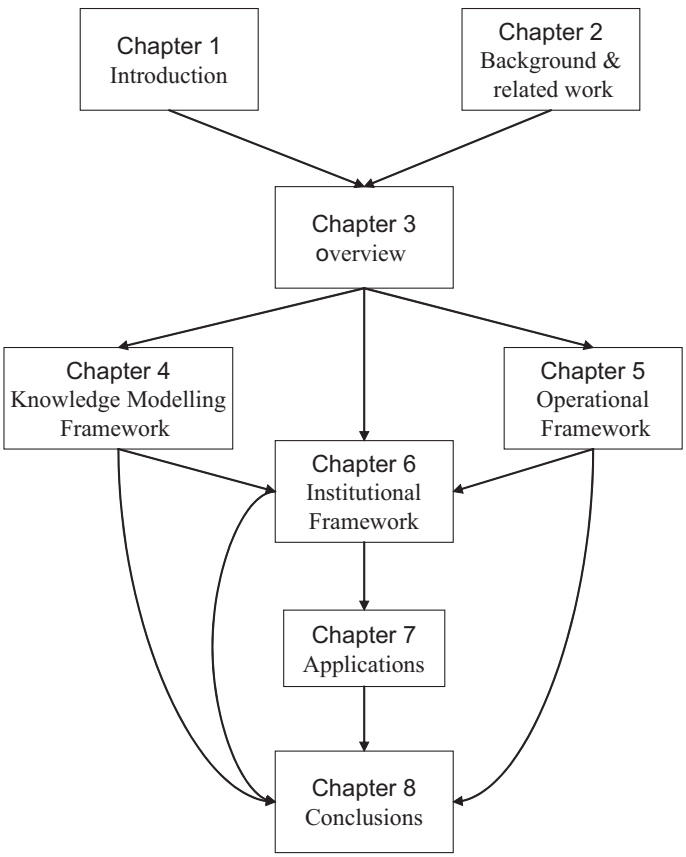


Figure 1.3: Thesis structure

Chapter 2

Background and related work

2.1 Introduction

Our work explores the potential of a Knowledge Modelling Framework for describing Multi-Agent Systems at the knowledge level, with the goal of enabling the Cooperative Problem Solving process to be adapted on-demand to fit the requirements of the problem at hand. Therefore, our review of related work has to deal with two main research areas: Knowledge Modelling (§2.2) and Multi-Agent Systems §2.4.

This background is completed with a section on Software Reuse (§2.3), and another section on Semantic Web Services (§2.5). On the one hand, the reason to include a section about software reuse is that we aim to maximize the reuse of agent capabilities across multiple domains. On the other hand, the reason to include a section devoted to Semantic Web Services (§2.5) is the shared interest of that field and Multi-Agent Systems on interoperation in open environments like the Internet. A more detailed organization of the chapter is sketched below:

- The chapter begins with a review of some Knowledge Modelling Frameworks in §2.2, focusing on those frameworks related with the Task-Method-Domain modelling paradigm: Generic Tasks (§2.2.1), Role-limiting Methods (§2.2.2), Components of Expertise (§2.2.3), KADS/CommonKADS (§2.2.4), UPML(§2.2.5) and current issues in knowledge modelling(§2.2.6). This section concludes with a subsection about reuse of problem solving methods (§2.2.6), and more specifically, it is mainly about the role of ontologies in component reuse.

- Section §2.3 is about those aspects of reuse that are more relevant to our work, and more specifically, those research lines assuming a compositional approach to software development, either explicitly or implicitly. Section §2.3.1 review the field of software libraries, which is similar to the idea

of libraries of problem solving methods in knowledge-engineering; Section §2.3.2 is about Component-Based Software Development (CBSD), which share some concepts with abstract architectures for knowledge-engineering; and finally, Section §2.3.3 deals with the requirements for a semantic-based reuse and the solutions.

- Section §2.4 addresses the wide field of Multi-Agent Systems. Although some dimensions and alternative views of Multi-Agent Systems will be presented, we favor an external, societal view of Multi-Agent Systems, thus avoiding topics such as agent architectures and agent theories. First, we review Cooperative Multi-Agent Systems (§2.4.1), including subsections on collaboration (§2.4.1), coordination (§2.4.1) and Cooperative Problem Solving (§2.4.1). The section continues with Team Formation (§6.5.3), a key activity of the Cooperative Problem-Solving process. Team Formation can be achieved by either a centralized task allocation (§2.4.2) and distributed task allocation (§2.4.2) approaches. Another section that is especially relevant for this thesis deals with agent interoperation in open environments (§2.4.3), including agent communication (§2.4.3), middle agents (§2.4.3), and matchmaking (§2.4.3). Section §2.4.3 ends with a review of infrastructures for developing and deploying Multi-Agent Systems in open environments (§2.4.3). Next section introduces social modelling approaches to Multi-Agent Systems (§2.4.4), and finally, the section on MAS concludes concludes with a review of agent-oriented methodologies (§2.4.5).

- The last section of this chapter is about Semantic Web Services (§2.5), and is mainly concerned with proposed frameworks (§2.5.1) and relevant work on composition and interoperation of Web services (§2.5.2).

At the end of the chapter we review the open issues and describe the type of problems that we expect to contribute to. In addition, the main differences between our approach and related work are discussed.

2.2 Knowledge Modelling Frameworks

There is much consensus that the process of building a knowledge system can be seen as a modelling activity. Building a knowledge system means building a computer interpretable model with the aim of making problem-solving capabilities comparable to a domain expert, but it is not intended to simulate the cognitive processes involved in human problem-solving.

In the early eighties the development of an expert system has been seen basically as a transfer process of human knowledge into an implemented knowledge base. This approach relies on the assumption than the knowledge is available there (in the expert) ready to be collected. Typically, this knowledge is acquired through interviewing experts, and the knowledge is implemented in some kind of production rules executed by some interpreter. But due to the limitation of one single knowledge representation formalism, and to the fact that

expert knowledge is often difficult to acquire (i.e. tacit knowledge) it was realized that the transfer approach adopted by first generation expert systems is only feasible for small prototypical systems, but it failed to produce large, reliable and maintainable knowledge bases. Therefore, with the introduction of the knowledge level [Newell, 1982] in the development of knowledge system, the knowledge acquisition phase is no longer seen as a transfer of knowledge, but as a model construction process [Clancey, 1989] with the following characteristics [Studer et al., 1998]:

- a model is an approximation of the reality, thus it is never ended;
- modelling is a cyclic process, so new observations can lead to a refinement, modification or completion of the already-made model; in the other side, the model may guide the further acquisition of knowledge; and
- modelling depends on subjective interpretations of the knowledge engineer, therefore this process is typically faulty.

Knowledge Modelling Frameworks propose methodologies, architectures and languages for analyzing, describing and developing knowledge systems. While different frameworks may differ on specific details, all of them are based on the idea of building a conceptual model of a system, which describes knowledge and inferences at a domain independent level. These frameworks have been influenced by the notion of the *knowledge level* [Newell, 1982], which proposes to describe a system by focusing on the knowledge they contain rather than the implementation structures of the knowledge (the *symbol level*). In addition, the knowledge level proposes to view a system as an agent with three classes of components: goals, actions and bodies of knowledge; and introduces a *principle of rationality* in the agent behavior: actions are selected to attain goals. Next, we will review the most prominent frameworks for knowledge engineering as a modelling activity at the knowledge level, namely: *Generic Tasks*, Role-Limiting Methods, Components of Expertise, KADS and CommonKADS, and UPML.

2.2.1 Generic Tasks

In the early eighties, the study of existing knowledge system for design and diagnosis evolved into the notion of a Generic Task (GT) [Chandrasekaran, 1986]. This approach proposes a task-oriented methodology for analyzing and building knowledge-systems. The main intuition underlying this proposal is that there are some recurring patterns in problem-solving activity (e.g. *hierarchical classification*, *abduction assembly* and *hypothesis matching*) that can be reused. Generic tasks are thus viewed as building-blocks that can be combined to build more complex problem-solving tasks (e.g. diagnosis) [Chandrasekaran, 1987]. This approach suggests that the representation of knowledge should closely follow its use (*strong interaction problem hypothesis* [Bylander and Chandrasekaran, 1988]), and that there are different organizations of knowledge suitable for different types of problem-solving: “Representing knowledge for the purpose of solving

some problem is strongly affected by the nature of the problem and by the inference strategy to be applied to the knowledge". A GT is characterized by:

- a generic description of its input and output;
- a fixed schema of knowledge types specifying the structure of the knowledge; and
- a fixed problem-solving method/strategy specifying the inference steps and the sequence in which these steps have to be carried out.

Since a GT fixes the type of knowledge required to solve a task, it provides a vocabulary that can be used to guide the knowledge acquisition process. A *Task Specific Architecture* (TSA) is an executable shell for a GT that provides a task-specific inference engine and a knowledge-base representation language. Therefore, a particular problem-solver is developed by instantiating a TSA with domain specific terms. However, this approach has the disadvantage of conflating the notion of task and problem-solving method. To overcome this limitation the notion of a *Task-Structure* was proposed [Chandrasekaran et al., 1992]: it makes a clear distinction between a task, which is used to refer to a type of problem, and a method, which is a particular way to accomplish a task. A task structure includes a set of alternative methods suitable for solving a task. A task can be decomposed into subtasks, thus a task structure is a hierarchical decomposition of tasks into subtasks, and the methods suitable for each task. The decomposition structure is refined to a level in which subtasks can be solved "directly" using available knowledge. From this point of view tasks refer to types of problems, while methods are specific ways of solving tasks [Chandrasekaran and Johnson, 1993].

2.2.2 Role-Limiting Methods

This approach [McDermott, 1988] focuses on the characterization of reusable Problem-Solving Methods. A Role-Limiting Method (RLM) is a method that declares the roles knowledge can play in that method. From this approach, whereas domain knowledge should not be acquired and represented with independence of the method, it is still explicitly and separately represented. An advantage of this approach is that method roles prescribe what knowledge should be acquired, therefore the expert only have to instantiating the generic roles with available knowledge. Furthermore, the problem-solving method facilitate explanations beyond a simple recall of inference steps as was usual in first generation expert systems. However, some limitations have been presented to this approach: first, knowledge-acquisition is completely driven by the RLM [Steels, 1990] and thus it is difficult to reuse domain-models for new RLMs; and second, RLMs have a fixed structure that is not well suited to deal with tasks that should be solved by a combination of several methods [Studer et al., 1998]. In order to overcome the inflexibility of RLMs, the concept of configurable RLMs has been proposed.

Configurable *Role-Limiting Methods* exploits the idea of a complex PSM being decomposed into subtasks, where each subtask may be solved by different methods [Poek and Gappa, 1993].

Role-limiting methods lead to a streamlined methodology for doing knowledge acquisition which have resulted in several tools that have been successfully applied in a variety of real-world applications.

2.2.3 Components of Expertise

Components of Expertise [Steels, 1990] is an attempt to synthesize the idea of Role-Limiting Methods and the task-structure analysis of the GT approach. In addition, this approach presents a *componential framework* that is expected to overcome some of the problems detected in previous work by imposing more modularity on the different components of expertise and emphasizing pragmatic constraints. The componential framework considers three classes of components to describe and build a problem solver: *tasks*, *models* and *Problem-Solving Methods*.

From a conceptual point of view, a task is characterized in terms of the type of problem to be solved (e.g. diagnosis, interpretation, design, planning, and so on). This characterization is based on properties of the input, output, and nature of the operations that map the input to the output. Usually, there is a main task that describes the application problem, but tasks can be decomposed into subtasks with input/output relations between them, resulting in a task structure [Chandrasekaran et al., 1992]. However, the pragmatic view focuses on task constrains that result from the environment or from the epistemological limitations of humans (models are limited in their accuracy and scope of prediction).

The componential framework addresses the question of knowledge modelling having in mind the separation of deep and surface knowledge [Steels, 1988]. From the perspective of deep expert systems, problem solving is viewed as a modelling activity in which some models of the world are constructed in order to solve a problem using that models. *Case models* are about the particular problem solving situation, which is determined by the task and methods at hand. However, *domain-models* are valid for a variety of cases. Domain-models describe domain specific knowledge that is used by Problem-Solving Methods to construct case models. There are two further subtypes of domain-models: *Expansion models* can be used to expand a case model by inference or data gathering; and *mapping models* are used to construct or modify a case model based on a mapping from elements of other models. Models can be constructed from different perspectives (for example, there are *functional models*, *causal models*, *behavioral models* and *structural models*) and represented in heterogeneous forms, like rules, hierarchies or networks.

Problem-Solving Methods (PSM) are responsible for applying domain knowledge to solve a task. A problem-solving method might decompose a task into subtasks or directly solve a subtask. In either case they can consult domain-models, create or change intermediary knowledge structures, perform actions to gather more data or expand a case model by adding or changing facts.

2.2.4 KADS and CommonKADS

KADS [Schreiber et al., 1993, Wielinga et al., 1993] is methodology for the analysis and design of knowledge system, which was further developed to *CommonKADS* [Schreiber et al., 1994a]. A basic characteristic of KADS is the construction of a collection of models, where each model captures specific aspects of the knowledge system to be developed as well as of its environment. In CommonKADS the *Organization Model*, the *Task Model*, the *Agent Model*, the *Communication Model*, the *Expertise Model* and the *Design Model* are distinguished:

- The *Organizational Model* describes the organizational structure in which the knowledge system will be introduced.
- The *Task Model* provides a hierarchical description of the tasks which are performed in the organizational unit, and the agents assigned to the different tasks.
- The *Agent Model* specifies the capabilities of each agent involved in the execution of tasks. In general an agent can be a human or some kind of software system.
- The *Communication Model* specifies the interactions between the different agents, including the type of information exchanged.
- The *Design Model* describes the system architecture, the representation and the computational mechanisms to realize a problem-solver according to the requirement of the target system captured at the Expertise Model and the Communication Model.
- The *Expertise Model* describes the knowledge required by agents to solve tasks, approaching knowledge modelling from three different perspectives: static, functional and dynamic. Accordingly, three layers for the expertise model are distinguished: *domain layer*, *inference layer* and *task layer*.
  - At the *domain layer* the domain specific knowledge required to solve tasks is modelled. Domain knowledge is conceptualized through domain-models. A domain-model provides a partial view on a part of the domain knowledge, which statements are conceptualized by a particular *ontology*. The main goal of structuring the domain layer is facilitate its reuse for solving different tasks.
  - At the *inference layer* the reasoning process of a knowledge system is specified following the Role-Limiting Methods approach [McDermott, 1988]. Problem-Solving Methods are described by primitive reasoning steps called *inference actions* as well as the *roles* played by domain knowledge. Dependencies between inference actions and roles are specified by an *inference structure*, which specifies how the

knowledge roles are used and produced by inference actions. Furthermore, the notion of roles provides a domain independent view of the domain

- The *task layer* provides a description of tasks in terms of input and output roles, and specifies the goals characterizing it. In addition, the task layer describes a decomposition of the task into subtasks, as well as the control flow over subtasks.

Both semi-formal and formal languages have been proposed to describe the Expertise Model: CML (Conceptual Modelling Language) [Schreiber et al., 1994b], which is a semi-formal language with a graphical notation, oriented towards the human understanding; and $(ML)^2$, which is a formal specification language based on first order predicate logic, meta-logic and dynamic logic.

The clear separation between the domain knowledge and the reasoning process at the inference and task layers enables two kind of reuse: on the one hand, a domain-model may be reused by different methods; on the other hand, a method may be reused in a different domain [Studer et al., 1998] by defining a new view on the domain. This approach weakens the *strong interaction problem hypothesis* [Bylander and Chandrasekaran, 1988], which is thus redefined as the *relative interaction hypothesis* [Schreiber et al., 1994a]: whereas some kind of dependency exists between the structure of the knowledge and the type of the task, it can be minimized by explicitly stating the dependencies between reasoning methods and domain knowledge [van Heijst, 1995]. Task and PSM ontologies may be defined as two viewpoints on an underlying domain ontology.

Since one of the goals of the CommonKADS approach is to facilitate reuse, a library of reusable and configurable components has been defined that can be used to build up an Expertise Model [Valente et al., 1994], including components to solve the following type of (generic) tasks: modelling, design, planning, assignment, prediction, monitoring, assessment and diagnosis [Breuker, 1994].

2.2.5 UPML

The *Unified Problem-solving Method Development Language* (UPML) [Fensel et al., 1999] is a framework for developing knowledge-intensive reasoning systems based on libraries of generic problem-solving components. UPML is a description language that integrates work on knowledge modelling, interoperability standards and ontologies. Rather than providing the object level description language, UPML provides an architectural framework that specifies the components, the connectors and a configuration of how the components should be connected (architectural constraints) to build a system. Moreover, design guidelines define a process model for building complex knowledge systems out of elementary components [Fensel and Motta, 2001].

The UPML architecture for describing a knowledge system consists of six different elements: tasks, domain-models, PSMs, ontologies, bridges, and refiners: tasks define the type of problems, PSMs specify the reasoning process, and

domain-models characterize the domain knowledge used by PSMs. Each of these elements is described independently to enable the reuse of task descriptions in different domains, the reuse of PSMs for different tasks and domain, and the reuse of domain knowledge for different tasks and PSMs. Ontologies provide the terminology used in tasks, PSMs and domain definitions. Again, this separation enables knowledge sharing and reuse. For example, different tasks or PSMs can share parts of the same vocabulary and definitions. A fifth element of a specification of a knowledge system are adapters, which are necessary to adjust the other (reusable) parts to each other and to the specific application problem. UPML provides two types of adapters: bridges and refiners. Bridges explicitly model the relationships between two distinguished parts of an architecture, e.g. between domain and task or between task and PSM. Refiners can be used to express the stepwise adaptation of elements of a specification, e.g. a task is refined into another more specialized task, or a PSM is refined into a more specialized PSM [Fensel, 1997b]. Again, separating generic and specific parts of the reasoning process maximizes reusability.

A very important role within the UPML framework is borne by ontologies. An ontology provides an explicit specification of a conceptualization, which can be shared by multiple reasoning components communicating during a Teamwork process. In UPML, ontologies are used to define the terminology and its properties used to define tasks, PSMs, and domain-models. UPML does not commit to a specific language style for defining a signature and its corresponding axioms. However, two styles of specifying signature and axioms are studied and has been proposed as suited formalisms: logic with sorts and a frame-based representation using concepts and attributes.

UPML systems are made by adapting and integrating components in a way constrained by the abstract architecture. The overall configuration process is guided by tasks that provide generic descriptions of problem classes. After selecting, combining and refining tasks they are connected with PSMs suitable for those tasks, and both tasks and PSMS are filled in with domain knowledge for the given domain.

2.2.6 Recent issues in knowledge modelling and reuse

The work on Problem-Solving Methods started in the eighties, when a number of researchers recognized common patterns in the reasoning processes of various knowledge systems and described them at a higher level of abstraction, decoupling them from the application domain. These conceptual models of the reasoning process make it easier to understand and maintain knowledge-based systems, and can be used to improve explanation facilities and reuse. The knowledge modelling community has carried out a large body of work on formalizing problem solving methods, on building libraries and on characterizing methods in terms of their assumptions and competencies. Research in this area is concerned with developing new PSMs, methodologies for PSM reuse, libraries of reusable PSMs, tools to support PSM development, and languages for representing PSMs. Recent developments of the knowledge mod-

elling community are undertaking the possibilities of Internet as a medium [Benjamins, 1997, Benjamins et al., 1999, Monica Crubezy and Musen, 2001].

A recent approach that is being addressed from both the software engineering and the knowledge engineering communities is that of software architectures [Shaw and Garlan, 1996, Garland and Perry, 1995]. The goal of *software architectures* is learning from system developing experience in order to provide the abstract recurring patterns for improving further system development. As such, software architectures contribution is mainly methodological in providing a way to specify systems. A software architecture has the following elements: (i) components, (ii) connectors, and (iii) a configuration of how the components should be connected [Garland and Perry, 1995]. Software architectures are designed to build applications by matching the specification of abstract components with the specification of components in a library or repository. Work on software architectures establishes an abstract level to describe the functionality and the structure of software artifacts, thus they are suited to describe the essence of large and complex software systems. Such architectures specify classes of application problems instead of focusing on the small and generic components from which a system is built up. The work on formalizing software architectures in terms of assumptions over the functionality of its components [Penix and Alexander, 1997, Penix, 1998, Penix and Alexander, 1999] shows strong similarities to recent work on PSMS, which define the competence in terms of assumptions over the domain knowledge [Benjamins et al., 1996c, Fensel and Straatman, 1996, Fensel and Benjamins, 1998a, Musen, 1998]. PSMs require specific types of domain knowledge and introduce specific restrictions on the tasks that can be solved by them. These requirements and restrictions are assumptions that play a key role in reusing Problem-Solving Methods, in acquiring domain knowledge, and in defining the problems that can be tackled by a knowledge-based system.

A complementary line of research is the work on ontologies to characterize consensual, formal and declarative knowledge models [Gruber, 1993a]. While Problem-Solving Methods describe the reasoning process of a knowledge-based system, ontologies provide the means to describe the domain knowledge that is used by these methods [Musen, 1998]. The availability of reusable methods and reusable domain knowledge reduces the development process of knowledge-based systems to a "plug-and-play" process [Walther et al., 1992].

Libraries of Problem-Solving Methods

Today, there exist several repositories or *libraries* of PSMs at different locations, with different scope, including the following: diagnosis [Benjamins, 1993], planning [Benjamins et al., 1996b], assessment [Valente and Lockenhoff, 1993], and design [Chandrasekaran, 1990].

All these libraries aim at facilitating the knowledge engineering process, yet they differ in various dimensions such as generality, formality, granularity and size. The type of a library is determined by its characterization in terms of these dimensions, and each library type is intended for a different role. For example, libraries of very generic PSMs (i.e. task neutral) aim to maximize

reuse, since they do not make any commitment to a particular task. However, at the same time, very general PSMs will require a considerable refinement and adaptation effort. This phenomenon is known as the *reusability-usability trade-off* [Klinker et al., 1991].

Another issue concerning component libraries is the way component specifications are organized and retrieved. There are several alternatives for organizing a library, and each of them has consequences for indexing PSMs and for their selection. Several researchers propose to organize libraries following a task-method decomposition structure [Chandrasekaran et al., 1992, Puerta et al., 1992, Steels, 1993, Shadbolt et al., 1993, Terpstra et al., 1993]. According to this organization structure, a task can be realized by several PSMs, each consisting of primitive or composite subtask that can again be realized by alternative methods. Guidelines for library design according to this principle have been discussed [Orsvarn, 1996], pointing out that PSMs are indexed according to two factors: the competence of the PSMs, and the assumptions under which they can be correctly applied. Selection of PSMs from such libraries should consider first the competence of the PSMs (selecting those whose competencies match the task at hand), and then the assumptions of PSMs (selecting those whose assumptions are satisfied).

Brokering of Problem Solving Methods

Problem-Solving Methods for knowledge systems establish the behavior of such systems by defining the roles in which domain knowledge is used and the ordering of inferences. Developers can compose PSMs that accomplish complex application tasks from primitive, reusable methods. The key steps in this development approach are task analysis, method selection from a library, and method configuration [Eriksson et al., 1995]. From the knowledge modelling community, this approach is described as a configuration process with the following activities: PSM selection according to their competence to solve a given task, verification of domain requirements, combining PSMs together, and mapping them to domain knowledge. This approach to software development is intended to support the engineer in the knowledge acquisition phase [Van de Velde, 1993] and to facilitate reuse [Fensel, 1997a]. The question of reuse has received a lot of attention last years from the knowledge modelling community [Benjamins et al., 1996a, Motta, 1999, Fensel and Motta, 2001]. Nowadays, the World-Wide Web is changing the nature of software development to a distributive *plug-and-play* process, which requires a new kind of managing software: intelligent software brokers. For selecting PSMs from a library, a broker needs to reason about characteristics of PSMs like their competence and their assumptions. A recent project, IBROW, [Benjamins et al., 1998, Benjamins et al., 1999], aims to provide an intelligent brokering service on the Web. One problem that appears while brokering components is the need to annotate components with semantic information (meta-data). From the knowledge-modelling approach, meta-data is provided by the three classes of components: PSMs, tasks and domain-models.

The IBROW approach to brokering libraries of problem-solving compo-

nents is that of configuring a knowledge-based system by selecting, adapting and integrating components retrieved from distributed libraries available on the Web [Monica Crubezy and Musen, 2001]. Essentially, the broker is a mediator between customers and providers of Problem-Solving Methods [Benjamins, 1997, Fensel and Benjamins, 1998b]. A customer is someone that has a complex problem but can provide domain knowledge that describes it and that supports problem-solving. The providers are developers of Problem-Solving Methods to be stored in libraries accessible through the Internet. PSMs are annotated with meta-data to support their selection process and invocation. The core of an intelligent broker for Problem-Solving Methods consists of an *ontologist* that supports the selection process of Problem-Solving Methods for a given application. Basically, such a broker has to provide support in building or reusing a domain ontology and in relating this ontology to an ontology that describes generic classes of application problems. This problem-type ontology has to be linked with PSM-specific ontologies that allow the selection of a method.

Ontology-based reuse

The knowledge modelling community has focused on ontologies [John H. Gennari and Musen, 1998, Studer et al., 1996] as a way to share consensual conceptualizations that are required to facilitate reuse. Ontologies are defined as "shared agreements about shared conceptualizations". Shared conceptualizations include conceptual frameworks for modelling domain knowledge; content-specific protocols for communicating among interoperating agents; and agreements about the representation of particular domain theories. In the knowledge sharing context, ontologies are specified in the form of definitions of representational vocabulary [Guarino, 1997b].

Although the definition of what ontologies are is still a debated issue [Guarino, 1997b], this term has achieve a considerably attention by the AI community, and in particular, it has been declared as a key issue in maximizing reuse [Fensel et al., 1997, Fensel, 1997a, Fensel and Benjamins, 1998b]. From that viewpoint, the main goal of an ontology is to facilitate knowledge sharing [Chandrasekaran et al., 1998]. In addition to enable reasoning about components in order to compare, retrieve, reuse or adapt them, ontologies play a central role in connecting software components by allowing the comparison of components using different vocabularies. The comparison of components described with different ontologies or different concepts of the same ontology is allowed by the annexion of ontology *mappings*.

Ontology mappings are declarative specifications of *matching relations* between ontologies, which consist of explicit specifications of the transformations required to match elements of one ontology to elements in another ontology. An example of a mapping is a *renaming*, but mapping can include any kind of syntactic or semantic transformation: *numerical mapping*, *lexical mapping*, *regular expression mapping* and others. In our framework the core components (tasks, capabilities and domain-models) are described with explicit, independent ontologies [Fensel et al., 1997]. In the context of modern componential frame-

works for knowledge systems development, ontology mappings are required to match problem requirements to tasks, Problem-Solving Methods to tasks, and domain-models to Problem-Solving Methods.

The importance of ontologies has even originated what has been called *ontology engineering* [Guarino, 1997b]. A remarkable outcome of this approach is that of reusing also the ontology-based connectors or *mappings*; not surprisingly, reuse of mappings is also improved by specifying a mapping ontology [Park et al., 1998].

2.2.7 Conclusions

Although there are some differences between the different frameworks presented above, a first conclusion of the review on Knowledge Modelling is that there exist much consensus about the use of three classes of components to model knowledge-based systems from a knowledge level approach [Fensel et al., 1999, McDermott, 1988, Chandrasekaran, 1986, Steels, 1990, Schreiber et al., 1994a]. This paradigm proposes three types of components: there are *tasks* describing problem types, *Problem-Solving Methods* (PSM) describing the reasoning steps required to solve a class of problems in a domain-independent way, and *domain-models* describing the properties of domain knowledge. We use the term *Task-Method-Domain* (TMD) frameworks to refer to this architectural pattern found in modern Knowledge Modelling Frameworks.

We are interested in TMD frameworks as a key to maximize reuse [Benjamins et al., 1996a, Motta, 1999, Fensel and Motta, 2001] and specifically, we are mainly interested in its application to the dynamic, on-demand configuration of Cooperative Multi-Agent Systems. Modern approaches from the knowledge engineering community are explicitly addressing reuse as an activity of selecting, configuring and assembling knowledge components from distributed libraries [Gennari and Tu, 1994, Eriksson et al., 1995, Fensel, 1997a, Fensel and Benjamins, 1998b, Benjamins et al., 1999]. TMD approaches envisage an scenario in which developers can compose Problem-Solving Methods that accomplish complex application tasks from primitive, reusable methods. This way of developing a knowledge-system is described as a configuration process (§2.2.6) and is intended to support the engineer in the knowledge acquisition phase [Van de Velde, 1993] and to facilitate reuse [Fensel, 1997a]. However, configuring such a system to build a completely new application is far away of becoming an automated process and is rather described as a semi-automated process [Gaspari et al., 1998, Gaspari et al., 1999, Benjamins et al., 1999, Penix, 1998]. Semi-automatic configuration can be used to support and assist knowledge engineers in the configuration and adaptation of knowledge-based systems [Eriksson et al., 1995, Tu et al., 1995, Studer et al., 1996, Fensel, 1997a, Fink, 1998, Penix and Alexander, 1997, Monica Crubezy and Musen, 2001], Such an approach to configuring a KBS has not been used to configure teams of problem solving-agents. The reason is probably the consideration of Team Formation as a runtime activity, whilst configuration in Knowledge Modelling Framework is considered as a knowledge engineering activity taking place on

design time.

This thesis shows how a TMD modelling framework can be used to guide the Team Formation process on runtime by fully automating the configuration of the team in terms of the competence required by a team to solve a problem. An automated configuration process is required to allow agent teams to be designed on-demand, according to the requirements of the specific problem at hand. Moreover, in addition to automating the configuration process, our thesis imposes that the problem specification should be done by the end-user, and not by a knowledge or software engineer.

Another keystone of TMD frameworks is the use of ontologies as explicit, declarative specifications of the conceptualizations used to characterize components. The use of ontologies to annotate components maximizes its reuse because enables the semantic comparison of components [Fink, 1998, Gaspari et al., 1999, Fensel and Benjamins, 1998b, Gaspari et al., 1998]. Consequently, we are including explicit ontologies to describe agent capabilities in a way that facilitates their reuse and configuration in the context of cooperative Multi Agent Systems.

2.3 Software reuse

The reuse of complete software developments and the processes used to create them have the potential to significantly ease the process of software engineering by providing a source of verified software artifacts [Wegner, 1984]. It is suggested than reuse of software artifacts can be achieved through the utilization of software libraries [Atkinson, 1997].

2.3.1 Software libraries

Essentially, a software library is a repository of information which can be used to construct software systems. The main goal of software libraries reuse is to enable previous development experiences to guide subsequent software development. Reuse have been partitioned in *compositional* and *generative* reuse. In particular, we are interested in compositional-approaches to software development [Biggerstaff and Perlis, 1989], which are characterized by the idea of selecting and composing existing components in order to achieve a desired system behavior.

An important aspect of compositional reuse is about the relationship between components in a software library. The are two major relationships: one relationship is the one between the client and supplier [Atkinson, 1997], where the definition of a component by a client refers to the existence of a component in the library as provided by its supplier; and the second relationship is that of inheritance, which allows the definition of one component to include the definitions of another. In general, there may be many relationships between two software components that can be used for retrieval from a software library: one component may be a subtype [Liskov and Wing, 1993] of another, be behaviorally

compatible [Smith, 1994] with another, or be substitutable [Duke et al., 1991] with another.

According to [Mili et al., 1995], there is an open problem of software composition called the *The Bottom Up Design Problem*, defined as:

> given a set of requirements, a set of components within a software library whose combined behavior satisfies the requirements.

The fundamental difficulty when considering this problem is how to decompose the requirements in such a way as to yield component specifications. A reverse approach is to search the space of all possible component compositions until one satisfying the requirements is found [Hall, 1993, Zhang, 2000]. Thus, composition of components can be regarded as composition of their specifications [Butler and Duke, 1998].

Although the use of a software library by a software engineer requires to know the processes of how to retrieve, insert and adapt components of the library, the issue of how to retrieve components is probably the central one, since the purpose of software libraries is to provide access to reusable verified components [Atkinson, 1997]. Component retrieval is defined as the process of locating the components that can be used in the construction of a particular application. From that view, retrieval is a process of obtaining a component in a library, $S \in \mathcal{L}$ satisfying a given query Q. A query Q imposes some requirements that should be compared with the specification of existing components $\mathcal{L}$ to check wether a particular retrieval criteria holds. There have been three classes of proposed solutions to this problem: *faceted* (classification), *signature-matching* (structural) and behavioral (functional) retrieval. These retrieval techniques can use many different indices as representations of components:

- *External Indices* seek to find relevant components based upon controlled vocabularies external to the component; including facets [Prieto-Daz, 1987], frames [Rosario and Ibrahim, 1994], lexical affinity [Maarek et al., 1991] and feature-based techniques [Börstler, 1995]

- *Static Indices* include type signature matching [Zaremski and Wing, 1995] and specification matching techniques [Rollins and Wing, 1991, Fischer et al., 1995, Zaremski and Wing, 1997], which seek to find relevant components based upon elements of the structure of components.

- *Dynamic Indices* seek to find relevant components by comparing input and output spaces of components, which are used by behavioral techniques [Hall, 1993, Mili et al., 1997]

2.3.2 Component-Based Software Development

Following the idea of reuse for component-based systems, Component-based software development (CBSD) focuses on building large software systems by integrating previously-existing software components. By enhancing the flexibility

and maintainability of systems, the ultimate goal is to reduce software development costs, assemble systems rapidly, and reduce the maintenance burden associated with the support and upgrade of large systems [Brown and Wallnau, 1996].

CBSD shifts the development emphasis from programming software to composing software systems [Clements, 1996], thus the notion of building a system by writing code is replaced by the notion of assembling and integrating existing software components. From the CBSD approach, constructing an application involves the use of prefabricated pieces, perhaps developed at different times, by different people and possibly with different purposes; therefore integrability of heterogeneous components is a key consideration when deciding whether to acquire, reuse, or build new components. In other words, software components can be deployed independently and are subject to composition by third parties [Szyperski, 1996].

Component capabilities and usages are specified by *interfaces*. From a compositional approach, an interface can be defined as a service abstraction, which defines the operations that the service supports independently from any particular implementation [Iribarne et al., 2002]. Interfaces can be defined using many different notations and representation languages. Usually component interfaces are described in three levels: signature level, semantic level and protocol level. Current approaches at the signature level use Interface Description Languages such as the ones defined by CORBA, COM and CCM. At the protocol level there are many interaction protocol description languages like those based in finite-state-machines [Yellin and Strom, 1997], Petri-Nets [Bastide et al., 1999], temporal logic [Han, 1999] or π-calculus [Canal et al., 2001]. At the semantic level the operational semantics of components are described using formal notations ranging from the Larch [Garland et al., 1993] family of languages based on pre-conditions and post-conditions to algebraic equations [Goguen et al., 1996] or refinement calculus [Mikhajlova, 1999].

2.3.3 Semantic-based reuse: ontologies

The informality of feature-based classification schemes for reuse is an impediment to formally verify the reusability of a software component. However, the use of formal specifications to verify reusability has associated a high reasoning cost, which jeopardizes the scalability of reusable software libraries. A way to increase the efficiency of formal specifications is to shift the overhead of formal reasoning from the retrieval to the classification phase of reuse [Penix et al., 1995]. This is done by using a classification scheme to reduce the number of specification matching proofs (usually some kind of implication) that are required to verify reusability. Components can be classified using semantic features that are derived from their formal specification. Retrieval can then be accomplished based on the stored feature sets, which allow an efficient verification of reusability relations [Penix and Alexander, 1999].

Following the philosophy underlying the *external indices* approach to software reuse and the semantic enrichment of component descriptions, it seems

natural to introduce ontologies as shared vocabularies to describe reusable components.

We agree with [Guarino, 1997b] about the potential role of explicit ontologies to support reuse, since ontologies can be used to enable semantic matching between components [Guarino, 1997a, Paolucci et al., 2002]. Therefore, semantic matching between an application specification and the components in a library can be used to verify a reusability relation. A key concept of ontology based reuse is that of mapping. Ontology mappings are declarative specifications of *matching relations*, which consist of explicit specifications of the transformations required to match elements of one ontology to elements in the other ontology. An example of a mapping is a *renaming*, but a mapping can include any kind of syntactic or semantic transformation, including *numerical mappings*, *lexical mappings*, *regular expression mappings* and others classes of mappings. An interesting property of ontology mappings are that mapping patterns themselves can become reusable components [Park et al., 1998].

2.3.4 Conclusions

An open issue of software reuse that is addressed within this thesis is the kind of language to be used for specifying components. Recent approaches remark the need for semantic information to describe software components, and there is an increasing consensus about the class of properties to describe a component; however, there are many differences among the object languages proposed to specify these properties, from simple keywords, to First-Order Logic. Since time is an important factor to take into account when considering the on-the-fly configuration of MAS, our goal is to achieve a trade-off between the expressive power of the object language and the computational efficiency of the inference mechanism (see §4.3.1).

2.4 Multi Agent Systems

Distributed Artificial Intelligence has historically been divided in two main areas [Bond and Gasser, 1988a]: Distributed Problem Solving (DPS) and Multi-Agent Systems (MAS). In the DPS approach, a problem is divided and distributed among a number of nodes which cooperate in solving the different parts of the problem. In the DPS model, the overall problem solving strategy is an integral part of the system. In contrast, MAS research is concerned with the behavior of a collection of possibly pre-existing autonomous agents aiming at solving a given problem [Jennings et al., 1998]. From that view, a MAS is a loosely coupled network of problem-solving entities that work together to find answers to problems that are beyond the individual capabilities or knowledge of the isolated entities [Durfee and Lesser, 1989]. More recently the term "Multi-Agent System" has come to a more general meaning, and it is now used to refer to all types of systems composed of multiple (semi-) autonomous components [Jennings et al., 1998]. The MAS approach advocates decomposing problems in

terms of autonomous agents that can engage in flexible, high level interactions, and this way of decomposing a problem aids the process of engineering complex systems [Jennings, 2000]. The characteristics of MAS are:

- each agent has incomplete information or capabilities for solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized, and
- computation is asynchronous.

Some reasons for the increasing interest in MAS research include: the ability to provide robustness and efficiency; the ability to allow inter-operation of existing legacy systems; and the ability to solve problems in which data, expertise, or control is distributed.

There are two main perspectives when approaching Multi Agent Systems: a macro or social level, focused on external, observable behavior, and a micro, or agent-oriented level, focused on the internal architecture of individual agents. We are more interested on the *macro* phenomena, rather than the *micro* phenomena; therefore, we are going to focus on the coordination and cooperation mechanisms of open agent societies, without paying much attention to neither agent theories nor agent architectures.

2.4.1 Cooperative Multi-Agent Systems

Cooperation is often presented as one of the key concepts which differentiates Multi-Agent Systems from other related disciplines such as distributed computing, object oriented systems, and expert systems [Doran et al., 1997]. However, the idea of cooperation in agent-based systems is yet unclear and sometimes inconsistent. There are many open questions like for example:

- What is cooperation? How does it relate to concepts like communication, coordination and negotiation?
- What sorts of cooperation are likely to be found in multi-agent systems? Which factors will affect cooperation strategies, and how?
- Is it meaningful to talk about reactive cooperation? Is cooperation a mentalistic, a behavioral notion or a mixture of the two?
- What are the key mechanisms and structures giving rise to cooperation?

The range of answers to these questions are many and varied, probably due to the different approaches adopted when addressing the cooperation issue; thus a typology of cooperation approaches seems necessary to help understand cooperation without offering a single definition of cooperation (Figure 2.1).

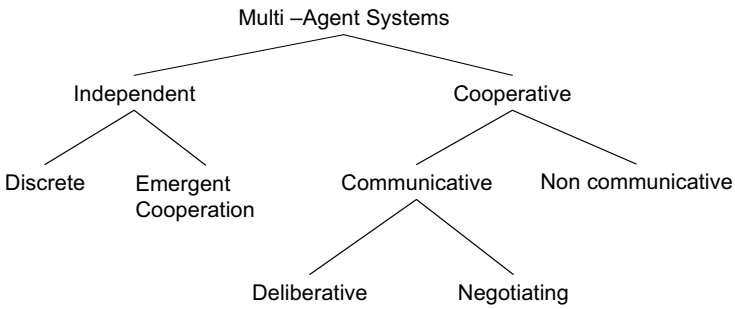


Figure 2.1: Cooperation typology

A Multi-Agent System is *independent* [Franklin and Graesser, 1996] if each agent pursues its own agenda independently of the others. A Multi-Agent System is *discrete* if it is independent and the agendas of the agents bear no relation to one another, thus there is no cooperation involved. However, cooperation is possible though agents have no intention of doing so, what can occur when cooperation is an *emergent* behavior resulting from the interaction of individuals (stigmergy is a good example [Beckers et al., 1994] of such a class of cooperation). On the other hand, there are systems in which the agendas (plans) of the agents include some way of cooperating with other agents. In *non-communicative* cooperation agents can coordinate by observing the behavior of the others [Franklin,]. In *communicative* systems agents can achieve coordination through the intentional sending and receiving of signals, which usually follow a *speech acts* style of communication. *Deliberative* agents jointly plan their actions so as to cooperate with each other. *Negotiating* agents are also deliberative, but the agents are basically competing, thus there is basically a different degree of self-interestingness.

There is an alternative viewpoint on cooperation that regards it as a property of the actions of the agents [Doran and Palmer, 1995]: cooperation occurs when agents have a (possibly implicit) goal in common (which no agent could achieve in isolation) and their actions tend to achieve that; or agents perform actions which enable or achieve not only their own goals but also the goals of other agents. This approach focuses on actions and goals, irrespective of how they arise. Therefore, from this view agents do not require to deliberate and goals may be implicit.

In contrast to the former approaches to cooperation that do not require intention, there is a more restrictive view on cooperation as a motivated activity. From that viewpoint, cooperation is defined as acting with others for a common purpose and a common benefit where the purpose should be motivated by an intention to act together [Norman, 1994]. That intention is usually referred as a commitment to joint activity [Bratman, 1992, Jennings, 1993]. This class of

cooperation relying on motivational attitudes is sometimes called collaboration [Wilsker, 1996, Grosz and Kraus, 1996] to differentiate it from other classes of cooperation.

Collaborative agents

Most early work in DAI dealt with a group of agents pursuing common goals [Lesser et al., 1989, Lesser, 1991, Durfee, 1988, Cammarata et al., 1983]. Agent interactions are guided by cooperation strategies meant to improve their collective performance, therefore early work on distributed planning took the approach of complete planning before action. These systems have to recognize, avoid or resolve dependencies or interactions between subproblems. For instance, [Georgeff, 1983] proposes a synchronizer agent to recognize and resolve such interactions.

Agents embedded in dynamic environments that are continuously sensing their environment and performing actions to change it are called *situated agents* [Rao et al., 1992]. These agents are resource-bounded [Bratman, 1988], they must reason and act under possibly stringent constrains on time and information. According to [Bratman, 1990], the intentions of the agent play a crucial role in such cases. Intentions can be seen as constrains on the deliberating and planning processes, hence reducing the reasoning effort. Systems based on mental attitudes like intentions are commonly called Belief-Desire-Intention (BDI) architectures [Bratman, 1988, Rao and Georgeff, 1991, Rao and Georgeff, 1995].

We review below three of the most influential contributions to the field, namely: Joint Intentions, SharedPlans and Planned Team Activity.

The *Joint Intentions* model [Cohen and Levesque, 1990, Levesque, 1990, Cohen and Levesque, 1991] represents one of the first attempts to establish a formal theory of multi-agent collaboration. This theory is a formal model of what motivates agent communication about teamwork. The basic premise rests in the idea of intention as the commitments to act in a certain mental state [Levesque, 1990]. A commitment represents a goal that persists over time. From this view, a team is composed of agents that jointly commit to the achievement of a team goal, called a *joint persistent goal* (JPG) [Cohen and Levesque, 1990]. An important conclusion of this theory is that by virtue of its joint commitments, an agent in a team has the responsibility to communicate private beliefs if it believes that the JPG is either achieved, unachievable or irrelevant [Cohen and Levesque, 1991]. Hence, the need of some form of communication is implicit in this model. Team goals are formed by an individual agent nominating a task as a proposed team goal, and communicating that intention until consensus is formed.

The *SharedPlans* [Grosz and Kraus, 1996, Grosz et al., 1999] model of collaboration emphasizes the need for a common high-level team model that allows agents to understand all requirements for plans to achieve a team goal [Grosz and Sidner, 1990], even if the individuals do not know the specific details of the collaborative plan or how to met the requirements The SharedPlans theory of collaboration is based on a rich view of plans. Rather than associating a

plan for some goal with a group of actions that can achieve it, a plan is instead a structure describing relationships between intentions (commitments) and information needs. Having a SharedPlan implies a joint mental state to do a group action: (1) mutual beliefs of a (partial) recipe; (2) individual intentions that the joint action will be carried over; (3) individual intentions that collaborators succeed in performing the constituent subactions; and (4) individual or collaborative plans for subplans. As a way of describing motivational attitudes, four different intention operators are introduced [Grosz and Kraus, 1993]: *intention-to*, *intention-that*, *potential-intention to* and *potential intention-that*. The first two are intentions that can be adopted by an agent, while potential intentions represent and agent's mental state when it is considering adopting and intention, but it is yet considering another possible courses of action. An *intention-to* perform some action represents an individual commitment on the part of an agent to perform that action, while an *intention-that* instead represents a commitment to certain states or conditions holding. Intentions-to serve a number of functions: (a) they constrain deliberations (an agent will seek ways to accomplish an intended action); (b) they represent commitments to action (an agent will not normally adopt new intentions that conflict with existing ones); and (c) agents monitor the success or failure of attempts to achieve an intention (failures can engender replanning). In contrast to an intention-to, an intention-that does not directly connote an action; rather, it implies that an agent will behave in a manner consistent with a collaborative effort, and can engender helpful behavior and also spawn monitoring actions. Communication requirements may arise from intentions-that, as opposed from being mandatory in the Joint Intentions model. If an agent has an intention-that about some group action to succeed then it will adopt a potential intention to perform any action it believes will help the group action to succeed. Therefore if an agent believes communicating some event or belief will aid in successfully prosecuting the group action, then it will be communicated.

The two former models have implicitly assumed that when agents establish either a Joint Commitment or a SharedPlan, they do so immediately and completely [Wilsker, 1996]; without any allowance for an intermediate mental attitude, like an expression of interest. Unlike the two previous theories, in the Planned Team Activity approach [Kinny et al., 1992, Sonenberg et al., 1994] is that plans to achieve some goal are supplied in advance, not generated by the agents, and that agents have complete knowledge of the full plans prior to joining a team. Therefore, agent behavior is bound and predictable, making this model advantageous in dynamic and real-time environments. On the other hand, teamwork is more brittle, since plans may fail within unpredictable environments. In addition, there is a greater responsibility on the agent designed, since the success of teamwork is tightly dependent on how well the plans are specified. The semantics of team's beliefs, goals and intentions are different from those in Joint Intentions. Specifically, the joint intentions of a team are expressed in terms of the joint intentions of its members, which reduce to single agent attitudes rather than by modal operators expressing shared attitudes. A team has a joint

intention towards a plan if: (1) every member has the joint intention towards the plan; (2) every member believes that the joint intention is held by the team; and (3) every member believes that all the members executing their respective individual plans results in the team executing the plan. This definition is similar to that of SharedPlans but without the intention that collaborators succeed. The process of Team Formation begins with an agent wishing to achieve some goal, but realizing that it is unable to do so by himself. The agent communicates with other potential participants by announcing the joint goal, joint plan and the individual roles to be assumed by each participant. An agent is capable of adopting a joint goal, a joint plan and a role within a plan if and only if: (a) has the necessary skills; (b) does not already believe the formula that needs to be adopted as a joint goal; (c) the preconditions of the plan are already believed by the team; (d) the joint goal is compatible with the current goals of the agent; and (e) the joint plan and role plan are compatible with the current intentions of the team member. Two strategies for Team Formation are considered by this proposal [Kinny et al., 1992]: (1) *commit-and-cancel*, and (2) *agree-and-execute*. In the commit-and-cancel strategy the team leader sends a request to each participant to "commit" to the joint goal, joint plan and role. If all the participants reply with a "committed" message within the permitted time the team has been formed; else the team leader sends a "cancel" message to them each agent committed and any team activity is abandoned. In the agree-and-execute strategy the team leader sends an "agree" to all participants, and if all reply affirmatively, sends an explicit request to all of them to execute the plan. Only at that point are the joint goal, joint plan and roles adopted by participants. Unlike the former strategy, no explicit message needs to be sent when an agent does not agree to participate, as the other agents have made no commitment yet. If a member is unable to achieve its goals within the team it has the responsibility to make other team members aware of its failure.

Another direction in multi-agent planning research is oriented towards modelling teamwork explicitly. This is particularly helpful in dynamic environments where team members may fail or where they may encounter new opportunities. For instance [Singh, 1994] proposes a family of logics for representing intentions, beliefs, knowledge, know-how, and communication in a branching-time framework. Whereas other theories are based exclusively on mental concepts, this approach combines mental and social concepts and proposes a formal theory of intentions for teams that considers the structure of teams explicitly, in terms of their members' commitments and coordination requirements [Singh, 1998]. Furthermore, this approach distinguishes between *exodeictic* (outward) and *endodeictic* (inward) intentions, which considers team structure. A team structure is defined by the constraints on the interactions —at the commitment and coordination levels— of its members.

Other works on collaboration based in multi-agent planning can be found for example in the *Social Plans* proposal [Rao et al., 1992] and in the Shared Planning and Activity Representation (SPAR) effort [Tate, 1998].

Coordination frameworks

Partial Global Planning (PGP) is a flexible approach to coordination that does not assume any particular distribution of sub-problems, expertise or other resources, but instead allows nodes to coordinate themselves dynamically [Durfee, 1988]. Cooperating agents adjust its own local planning so that the common planning goals are met, and communicate its plan to others to improve predictability and network coherence. The PGP approach to distributed coordination improved the coordination of agents in a network by scheduling the timely generation of partial results, avoiding redundant activities, shifting tasks to idle nodes, and indicating compatibility between goals. Identifying and generalizing the types of coordination relationships that were used by the basic PGP algorithm has lead to the Generalized PGP (GPGP) [Decker and Lesser, 1992]. *Generalized Partial Global Planning*(GPGP) is a coordination algorithm described in a modular, domain independent way, which can be tuned for particular intra-task environment behaviors(primarily the creation and refinement of local scheduling constraints). GPGP extends (as well as generalizes) the PGP algorithm along two lines: handling real-time deadlines and improving the distributed search among schedulers. GPGP can be seen as an extendable family of coordination mechanisms that form a basic set of mechanisms for teams of cooperative autonomous agents [Decker and Lesser, 1995]. This approach provides a set of modular coordination mechanisms; a general specification of these mechanisms involving the detection and response to certain abstract *coordination relationships* (not tied to a particular domain); and a more clear separation of the coordination mechanisms from an agent's local scheduler that allows each to better do the job for which it was designed.

TAEMS [Decker, 1996] was designed as a modelling language for describing the task structures of agents, supporting the GPGP approach to coordinated agent behavior. The acronym stands for Task Analysis, Environmental Modelling and Simulation. A TAEMS task structure is essentially an annotated task decomposition tree (actually a graph). The highest level nodes in the tree, called task groups, represent goals that an agent may try to achieve. Below a task group there will be a sequence of tasks and methods which describe how that task group may be performed. Tasks represent sub-goals, which can be further decomposed in the same manner. Methods, on the other hand, are terminal, and represent the primitive actions an agent can perform. Annotations on a task describe how its subtasks may be combined to satisfy it. Another form of annotation, called an interrelationship, describes how the execution of a method, or achievement of a goal, will affect other nodes in the structure. The TAEMS framework is designed to handle issues of real-time (e.g. scheduling to deadlines) and meta-control (e.g. to avoid the need of detailed planning at all possible node interactions). Much of what is represented in TAEMS structures is also quantitatively described, including expected execution characteristics, resource usage and specific ways to derive the quality of a task from the qualities of the combined subtasks. These quantitative aspects allow the agent to compare and contrast possible plans, predict their effects, and reason about the need for

coordination with other agents.

TEAMCORE [Tambe, 1997, Pynadath et al., 1999, Tambe et al., 2000] is an agent architecture that integrates many of the basic principles of the joint intentions theory and the Shared Plans approach. This is a perspective based on an explicit, domain independent model of teamwork that has include learning [Tambe et al., 1999] as the most remarkable issue.

The Cooperative Problem-Solving process

A general framework for the Cooperative Problem-Solving process has been described in [Wooldridge and Jennings, 1999], with four stages: recognition (an agent identifies the potential for cooperation), Team Formation, plan formation (collective attempts to construct an agreed plan) and execution. The authors adopts an internal (endodeictic) perspective, the approach is to characterize the mental states of the agents that leads them to solicit and take part in cooperative action. The model is formalized by expressing it as a theory in a quantified *multi-modal logic* . Starts from the following desiderata: agents are autonomous, cooperation can fail, communication is essential, communicative acts are characterized by their effects, agents initiate social processes, are mutually supportive and are reactive.

The view on Multi-Agent Systems as decoupled networks of autonomous entities is usually associated to a distributed model of expertise, regarded as a collection of specialized agents with complementary skills. Thus *team selection* is defined as the process of selecting a group of agents that have complimentary skills to achieve a given goal [Tidhar et al., 1996].

2.4.2 Team Formation

Team Formation is defined as the process of selecting a group of agents that have complimentary skills to achieve a given goal [Tidhar et al., 1996]. Most approaches to team selection view the process of achieving team goals as means-end analysis [Bratman et al., 1991, Rao, 1994] with two steps: first, selecting a group of agents that will attempt to achieve a goal [Levesque, 1990, Cohen and Levesque, 1991, Wooldridge and Jennings, 1994, Rao and Georgeff, 1995]; and second, selecting a combination of actions that agents must perform to achieve the goal [Grosz and Kraus, 1996, Tambe, 1997, Wooldridge and Jennings, 1999]. This combination of actions is typically described as a sequence of actions or a plan, like in in the Social Plans approach [Rao et al., 1992], the Planned Team Activity model [Kinny et al., 1992, Sonenberg et al., 1994], and the SPAR proposal (Shared Planning and Activity Representation) [Tate, 1998].

There are several approaches and variations over this basic schema. One of the more extended approach is to use plans as "recipes" [Georgeff and Lansky, 1987, Bratman et al., 1991, Sonenberg et al., 1994, Tidhar et al., 1996]. Since plans are provided by the user at compile time

the process of planning in the classical sense is unnecessary and can lead to significantly better performance.

Centralized Task allocation

One of the first methods for selecting agents for cooperative action was the *contract-net* protocol [Smith, 1940]. Given a task to perform, an agent determines whether the task can be decomposed into subtasks and announce these tasks to other agents by sending a "*call for proposals*". Bidders can reply with a bid to perform a task, indicating how well (price, quality, time, etc.) can they perform it and, finally, the contractor collects the bids and awards the task to the best bidder. This protocol enables dynamic task allocation, allows agents to bid for multiple tasks at a time, and provides natural load balancing [Jennings et al., 1998]. This protocol has however some limitations, like the absence of conflict detection and resolution, the impossibility for agents to refuse bids, or the absence of pre-emption in task execution [Jennings et al., 1998]. Some extensions of the protocol have been proposed to rectify some of its shortcomings [Sandholm, 1993]. The contract net protocol has been so extensively used, modified and extended by researchers in the field of multi-agent coordination [Sandholm, 1993, Dignum et al., 2001], that it has been included in the standardization effort carried out by the Foundation for Intelligent Physical Agents (FIPA) [FIPA, 2002]. However, the suitability of the Contract Net protocol for open MAS is under analysis, as it seems to be very dependent on the value of the deadline used when waiting for bids, and in the number of agents [Juhasz and Paul, 2002]. Such unguided team selection involves an exponential number of possible team combinations, and a blow-out in the number of interactions required to select the members of a team. Some attempts to overcome these problems that still rely on some kind of global plan employ problem requirements to guide the team selection and reduce the number of possible teams [Tidhar et al., 1996].

Distributed Task Allocation

Anytime algorithms with low ratio bounds have been proposed based on distributed coalition formation algorithms [Shehory et al., 1997, Shehory and Kraus, 1998]. A coalition is defined as "a group of agents who have decided to cooperate in order to achieve a common goal". Given a set of tasks and agents, these algorithms search a combination of coalitions (overlapping or not) to solve each task, taking into account the agents limited resources.

Distributed task allocation methods are appropriate for DPS and cooperative MAS [Sycara et al., 1996], since agents cooperate to increase the overall outcome of the system. Non super-additive environments (in a super-additive environment any combination of two groups of agents into a new group is beneficial) consider also task dependencies (task precedence and competing resources). Other examples of distributed planning propose methods for coordinating plans

at abstract levels [Clement and Durfee, 1999] by using information about how abstract plans can be refined in order to identify and avoid potential conflicts.

In the *Team Formation by dialog* approach [Dignum et al., 2001] autonomous agents are able to discuss the Team Formation, using structured dialogues, with an emphasis on persuasion. Follows the four stages model of the Cooperative Problem-Solving process drawn in [Wooldridge and Jennings, 1999]. This approach is based on the Dialogue theory [Walton and Krabbe, 1995], that proposes to structure dialogues by rules, so dialogues are not completely free neither completely fixed. The initial situation of negotiation is a conflict of interests, together with a need for cooperation, where the main goal is to make a deal. This theory adopts an internal view on agents based on a BDI model and an architecture containing reasoning, planning, communication and social reasoning modules. The initiator agent makes a partial plan for the achievement of a goal and looks for potential teams based on abilities (static), opportunities (situational), and willingness.

2.4.3 Interoperation in open environments

One of the current factors fostering MAS development is the increasing popularity of the Internet, which provides the basis for an open environment where agents interact with each other to reach their individual or shared goals. To successfully communicate in such an environment, agents need to overcome two fundamental problems: first, they must be able to *find* each other (since agents might appear or disappear at any time), and once they have done that, they must be able to *interact* [Jennings et al., 1998].

Interaction is one of the most important features of an agent [Nwana and Woolridge, 1996]. It is in the nature of agents to interact to share information, knowledge and goals to achieve. Three key elements have been outlined for a successful interaction:

- A common agent communication language and protocol
- A common format for the content of communication
- A shared ontology

Furthermore, in open environment there is a need of mechanisms for advertising, finding, using, combining, managing and updating agent services and information [Decker et al., 1997b]. Next follow a very brief section on agent communication topics and a review of middle agents as one way of implementing the aforementioned requirements.

Agent Communication

Agents must share a communication language to be able to interoperate. There are two main approaches to agent communication languages [Genesereth and Ketchpel, 1997]: in the *procedural* approach communication

is based on executable content, which can be accomplished by using programming and scripting languages, e.g. Tcl [Ousterhout, 1990]. Since procedural languages are difficult to control, coordinate and merge, declarative languages are preferred for the design of agent communication languages, specially in open environments. Most declarative communication languages [FIPA, 2003, Finin et al., 1994, Labrou and Finin, 1997] are based on the *speech acts* theory [Searle, 1969]. For this approach, communication is modelled through illocutionary acts called *performatives* (e.g. request, inform, agree), which are conceptualized as actions intending to produce some effect on the receiver, like performing some task (request) or giving some information (query). Although such performatives can characterize message types, efficient languages for expressing message content so as to allow a meaningful communication, have not been effectively demonstrated [Jennings et al., 1998]; thus the problem of representing and sharing meaning through ontologies is still open [Gruber, 1993b].

Middle agents

A common approach to overcome the interoperability problems agents face in open environments is the introduction of a middleware layer between requesters and providers of services and the use of a shared language and ontologies for describing both the tasks to be solved and the capabilities available. Having a mediation service is very useful since problem solving agents can advertise their capabilities, and the requester may look for agents with the capabilities more appropriate for the problem at hand. Usually, the mediation layer is realized by *middle agents* [Decker et al., 1997b] specialized in reasoning about and supporting the activities of other agents.

Many ideas about middle agents have precedents in the work on *mediators*. The notion of mediators was initially proposed in the field of Information Systems. A foundational paper is [Wiederhold, 1992], which introduces *mediators* as a technique to handle large-scale information systems in open and distributed environments. Thus, it is not strange that current ideas on middle agents were initially applied in the field of Intelligent Information Integration [Wiederhold, 1993] and Information Brokering [Jeusfeld and Papazoglou, 1996, Martin et al., 1997] as a way to locate and combine information coming from multiple and heterogeneous sources, e.g. relational and object-oriented databases. The notion of mediators is also studied as a software pattern [Rising, 2000] and is used in multi-layer information architectures [Wiederhold and Genesereth, 1997]. Mediators and brokers have been originally conceived as providing an added-value services for information-based applications. A middle agent can be seen as a mediator between requesters and providers of services, in which the information being mediated is constituted by the description of available services. Some introductions to middle agents can be found in [Decker et al., 1996] and [Decker et al., 1997b], and a taxonomy of middle agents appears in [Wong and Sycara, 2000]. Below follows a brief review of approaches to middle agents, though there is not clear differentiation between

the different types considered.

- *Facilitators* are agents to which other agents surrender their autonomy in exchange of the facilitator's services [Erickson, 1996a, Genesereth and Ketchpel, 1997]. Facilitators can coordinate agents' activities and can satisfy requests on behalf of their subordinated agents.
- *Mediators* are agents that exploit encoded knowledge to create services for a higher level of applications [Wiederhold, 1992]. For a detailed account of the differences between mediators and facilitators see [Wiederhold and Genesereth, 1997].
- *Matchmakers* and *yellow pages* assist service requesters to find service providers based on advertised capabilities. Services found that matches a given request are communicated to the requester, thus it must choose and contact the selected provider directly. [Decker et al., 1996].
- *Brokers* are agents that receive requests and are able to contacting with appropriate providers on behave of the requester. Thus, tasks are delegated to brokers that locate and communicate with suitable by themselves, freeing the requester of knowing the details required to communicate with a specific provider. The difference between brokers and matchmakers is that the matchmaker only introduces matching agents to each other, whereas a broker handles all the communication with the capability providers [Decker et al., 1996].
- *Blackboards* are repository agents that receive and hold requests for other agents to process [Nii, 1989].

Preliminary experiments [Decker et al., 1997b] shows that each type of middle agent have its own performance characteristics and is best suited for a certain type of environment. For example, while brokered architectures ar more vulnerable to failures, they are also able to cope more quickly with a rapidly fluctuating agent work-force. A general problem with the existing systems is that they do not overcome the gap between *push* and *pull* access to information [Haustein and Ludecke, 2000], since they commit to only one access model. There is, however, a considerable interest in combining both ways of accessing information. By caching data from a "push" source, a combination of a mediator and a broker could solve the access mismatch problem, providing both access modes.

Matchmaking and Agent Capability Description Languages

Typically, the function of middle agents is to match service-requests with service providers, where services are provided by agents. To enable matchmaking, both providers and requesters should share a common language to describe both service-requests and agent capabilities, which is called an Agent Capability Description Language (ACDL) [Sycara et al., 2001] (called also an Agent Service

Description Language, due to the quite usual view on agent capabilities as "services" provided to clients).

Matchmaking is the process of finding an appropriate provider of capabilities (or services) for a requester [Sycara et al., 1999a, Sycara et al., 1999b]. Some ACDLs supporting matchmaking are reviewed below, namely: the Logical Deduction Language (LDL++), the Interagent Communication Language (ICL), the Language for Advertisement and Request for Knowledge Sharing (LARKS), and the DARPA Agent Markup Language (DAML-S).

- LDL++ is a logical deduction language similar to Prolog that is used by brokers in the Infosleuth [Nodine et al., 1999] distributed agent architecture. LDL++ supports inferences about whether an expression of requirements matches a set of advertised capabilities.

- ICL is the interface, communication, and task coordination language shared by OAA agents, regardless of what platform they run or on what computer language they are programmed in [Martin et al., 1999, Cheyer and Martin, 2001]. OAA agents employ ICL to perform queries, execute actions, exchange information, and manipulate data in the agent community. ICL includes a layer of conversational protocols (such as KQML or FIPA), and a content layer. The content layer has been designed as an extension of PROLOG, to take advantage of unification and other features of PROLOG. Every agent participating in an OAA-based system defines and publishes its capabilities expressed in ICL. These declarations are used by a facilitator to communicate with the agent and also for delegating service requests to the agent.

- LARKS [Sycara et al., 2002] (Language for Advertisement and Request for Knowledge Sharing) is a language used by matchmaking agents to pair service-requesting agents with service-providing agents that meet the requesting agents [Sycara et al., 1999a], and is used by agents in the RETSINA [Sycara et al., 2001] agent infrastructure. When a service-providing agent registers a description of its capabilities with a middle agent, it is stored as an "advertisement" and added to the middle agent's database. Therefore, when an agent inputs a request for services, the middle agent searches its database of advertisements for a service-providing agent that can fill such a request. Requests are filled when the provider's advertisement is sufficiently similar to the description of the requested service. LARKS is capable of supporting inferences. It also incorporates application domain knowledge in agent advertisements and requests. Domain-specific knowledge is specified as local ontologies in the concept language ITL.

- ATLAS (Agent Transaction language for Advertising Services) is a DAML-based agent advertising language that will enable agents and devices to locate each other and interoperate [Paolucci et al., 2002]. ATLAS is based in DAML-S, an ontology to annotate Web Services with semantic information

[The DAML-S Consortium, 2001]. The DAML-S ontology uses concepts that are similar to the purpose and the requirements of an Agent Capability Description Language. Therefore, we find very similar elements in both ACDLs and Semantic Web Services description languages. Specifically, the view of the DAML-S consortium is that DAML-S descriptions are used by agents in support of the automated discovery, interoperation, composition, execution and monitoring of services. A matchmaker for DAML-S matchmaking has been proposed that utilizes two separate filters: one compares *Functional Attributes* to determine the applicability of advertisements, and the other compares *Services Functionalities*. *Subsumption* is the inference operation used to determine if two specifications match.

There are other proposals for ACDLs, based upon some extension of Petri Nets, like *Possibilistic Petri Nets* [Jonathan Lee and Chiang, 2002], the object-based extension called *G-Net* [Xu and Shatz, 2001] and the constraint-based model of *fitness-for-purpose* [White and Sleeman, 1999], among others.

It is also interesting to review other research on capability descriptions not specifically designed to describe agent capabilities, although they can be adapted for that purpose, like skills modelling in human organizations [Stader and Macintosh, 1999], capability descriptions for PSMs [Aitken et al., 1998], or process/action modelling techniques such as SPAR [Tate, 1998] and ADL [Pednault, 1989]. See [Wickler and Tate, 1999] for a wide survey on capability description for software agents.

Agent infrastructures for Cooperative Problem-Solving

There are several architectures and standards focused on open agent architectures and mechanisms to achieve interoperability. These infrastructures are based on some notion of mediation or middle agents: like *yellow-pages* in the FIPA abstract architecture , *matchmakers* in Retsina, *brokers* in OAA and task-planning agents in UMDL. However, there are some architectures oriented towards industrial applications, e.g. GRATE and ARCHON.

- *FIPA*[1] has produced a collection of specifications which aim is to become an standard for the interoperability of heterogeneous software agents. The standardization effort includes an *Abstract Architecture* dealing with the abstract entities that are required to build agent services and an agent environment. A FIPA platform contains a communication channel, an Agent Name Server (ANS) that is used as a "white pages" service, and a Directory Facilitator(DF), which acts as a "yellow pages" service.

- *UMDL*[2] provides a distributed architecture [Birmingham et al., 1995] for a digital library that can continually reconfigure itself as users, contents, and services come and go. This has been achieved by the development

[1]FIPA stands for the Foundation for Intelligent Physical Agents
[2]University of Michigan Digital Library.

of a multi-agent infrastructure with agents that buy and sell services from each other by using commerce and communication services/protocols [Vidal et al., 1998], that is called the Service Market Society (SMS). The SMS allows for the decentralized configuration of an extensible set of users and services [Durfee et al., 1998]. There are many types of agents in the UMDL agent architecture: there are information agents specialized in complementary knowledge areas; there are user interface agents that support the user in specifying queries; and there are also task planning agents [Vidal and Durfee, 1995] that are able to perform matchmaking between queries and agent services. Services are described in Loom.

- *RETSINA*[3] is an open multi-agent architecture that supports communities of heterogeneous agents [Sycara et al., 2001]. Distributed approach to information and problem-solving tasks (search, gathering, filtering, fusion, etc). The RETSINA system has been implemented on the premise that agents in a system should form a community of peers that engage in peer to peer interactions. Any coordination structure in the community of agents should emerge from the relations between agents, rather than as a result of the imposed constraints of the infrastructure itself. In accordance with this premise, RETSINA does not employ centralized control within the MAS; rather, it implements distributed infrastructure services that facilitate the interactions between agents, as opposed to managing them.

- *OAA*[4] [Cheyer and Martin, 2001] is a framework for building flexible, dynamic communities of distributed software agents. OAA enables a cooperative computing style wherein members of an agent community work together to perform computation, retrieve information, and serve user interaction tasks. Communication and cooperation between agents are brokered by one or more facilitators, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents [Martin et al., 1999].

- *DECAF* (Distributed, Environment-Centered Agent Framework) is a toolkit which allows a principled software engineering approach to building Multi-Agent Systems. The toolkit provides a platform to design, develop, and execute agents. DECAF provides the necessary architectural services of a large-grained intelligent agent: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis . This is essentially, the internal "operating system" of a software agent, to which application programmers have strictly limited access. The control or programming of DECAF agents is provided via a GUI called the Plan-Editor. In the Plan-Editor, executable actions are treated as basic building blocks which can be chained together to achieve a more complex goals in the style of a *Hierarchical Task Network*. This issue provides

[3]Reusable Environment for Task-Structured Intelligent Networked Agents.

[4]OAA stands for the Open Agent Architecture, developed at the SRI International's Artificial Intelligence Center (AIC)

a software component-style programming interface with desirable properties such as component reuse (eventually, automated via the planner) and some design-time *error-checking*. The chaining of activities can involve traditional *looping* and *if-then-else* constructs. This part of DECAF is an extension of the RETSINA and TAEMS task structure frameworks. Unlike traditional software engineering, each action can also have attached to it a performance profile which is then used and updated internally by DECAF to provide real-time local scheduling services . The reuse of common agent behaviors is thus increased, since the execution of agent behaviors does not depend only on the specific construction of the task network but also on the dynamic environment in which the agent is operating. Furthermore, this model allows for a certain level of non-determinism in the use of the agent action building-blocks.

GRATE [Jennings et al., 1992] is a general framework which enables to construct MAS for the domain of industrial process control. Embodies in-built knowledge related to cooperation, situation assessment and control. Designer can utilize this knowledge (reuse , configuration of preexisting knowledge) and augment it with domain specific information, rather than starting from scratch. More focused on agent architecture than MAS architecture. The in-built knowledge is represented by generic rules encoding sequences of actions. ARCHON [Wittig et al., 1994] is an extension of GRATE [Jennings et al., 1992] with reactive mechanisms. Many of GRATE's generic rules encode sequences of actions resulting in common patterns of rule firing, these patterns can be grouped into units of activity similar in nature to reactive planning systems (precompiled plans that behaves like in an unplanned, reactive manner). The result is a hybrid approach in which both general rules and reactive mechanisms are combined. ARCHON concentrates upon loose coupling of semiautonomous agents. There is no representation of an overall goal, but only goals of the agents that together met the overall goals of the community. ARCHON has been applied to pre-existing computational systems, although its concepts may well be used as enhancements to more conventional (e.g. client/server) integration architectures.

2.4.4 Social approaches

There are some aspects of complex system development that become more difficult by adopting an agent-based approach [Jennings, 2000]. Since agents are autonomous, the patterns and the effects of their interactions are uncertain, and it is extremely difficult to predict the behavior of the overall system based on its constituent components, because of the strong possibility of emergent behavior. These problems can be circumvented by imposing rigid and preset organizational structures, but these restrictions also limit the power of the agent-based approach. As an answer to these difficulties a *social level* view has been proposed [Jennings and Campos, 1997] that takes the *knowledge level* [Newell, 1982] anal-

ysis approach as a starting point. Whereas the knowledge level view stripped away implementation and application specific details from problem solvers, the social level view focuses on the organizational aspects of agent societies with the primary goal of analyzing system behaviors abstracted from implementation details or specific interaction protocols [Jennings and Campos, 1997]. The GAIA methodology [Wooldridge et al., 2000] follows this approach which allows to describe agent-based systems as computational organizations that are defined in terms of roles, interactions and obligations.

The *Civil Agent Societies* (CAS) [Dellarocas, 2000] is a framework for developing agent organizations which follows the metaphor of civil human societies based on social contracts, and is oriented towards marketplaces and B2B e-commerce. The CAS approach uses the Contract Net interaction protocol, social norms, notary services and exception handling services.

Another social approach to Multi-Agent Systems is described in [Panzarasa and Jennings, 2001] that is based on a conception of cognition, both at the individual and the collective level, and examined in relation to contemporary organization theory. Yet another organization-oriented model for agent societies is found [Dignum et al., 2002].

Another view of open agent organizations is that of *electronic institutions* as a metaphor of human institutions. A electronic institution is a virtual place where agents meet and interact according to the communication policies and norms defined by the institution. Formalization of electronic institutions [Esteva et al., 2001] underpins the use of structured design techniques and formal analysis, and facilitates development, composition and reuse. ISLANDER is a formal language with a graphical representation that allows to define [Esteva et al., 2002a] and verify [Huguet et al., 2002] the specification of an electronic institutions. Another advantage of such a formal language is that, given the specification of an electronic institution, it is possible to generate skeletons for the development of agents for that institution [Vasconcelos et al., 2001].

2.4.5 Agent-Oriented Methodologies

Agent technology has received a great deal of attention in the last few years and is also beginning to attract the industry. But in spite of the extensive research and successful application of agent theories, languages and architectures, there is little work for specifying techniques and methodologies to develop agent-based applications. Furthermore, the usual approach to the development of agent-oriented methodologies have been to adapt or extend an existing methodology to deal with the relevant aspects of agent-oriented programming [Iglesias et al., 1998]. These extensions have been carried out mainly in three areas: *object-oriented* methodologies, *software engineering* and *knowledge-engineering*.

Extensions of Object-Oriented Methodologies

There are several reasons to use object-oriented methodologies as the basis for an agent-oriented methodology: (1) there are many similarities between both paradigms [Burmeister, 1996, Kinny and Georgeff, 1996], and specifically, there is a close relationship between DAI and object-based concurrent programming [Bond and Gasser, 1988b, Gasser and Briot, 1992, Yoav Shoham, 1993]; and (2) there is a considerable experience is using object-oriented languages to implement agent-based systems.

Some examples of agent-oriented methodologies based on OOP are the following: Agent-Oriented Analysis and Design [Burmeister, 1996], Agent Modelling Technique for Systems of BDI agents [Kinny and Georgeff, 1996], Multi-Agent Scenario-Based Method (MASB) [Moulin and Brassard, 1996] and Agent Oriented Methodology for Enterprize Modelling [Kendall et al., 1995]

However, there are some aspect of agents not addressed by object oriented methodologies [Burmeister, 1996, Yoav Shoham, 1993, Kendall et al., 1995]: (1) the agent style of communication can be much more complex than the method invocation style in OOP; (2) agents can be characterized by their mental state; and (3) agents can include a social dimension not existing in OOP.

Extensions of Knowledge-Engineering Methodologies

Knowledge engineering methodologies can provide a good basis for MAS modelling by exploiting a human inspired style of problem solving. Since agents have cognitive features, the experience achieved in *knowledge acquisition* and *knowledge modelling* methodologies can be applied to agent development. In addition, existing tools and libraries of Problem-Solving Methods [Breuker and Van de Velde, 1994] can be reused.

CoMoMAS [Glaser, 1996] is an extension of CommonKADS [Schreiber et al., 1994a] for MAS modelling. The following models are defined:

- The *Agent Model* defines the agent architecture and the agent knowledge, that is classified as social, cooperative, control, cognitive or reactive knowledge.
- The *Expertise Model* defines the cognitive and reactive competencies of agents, distinguishing between tasks, problem solving (PSM) and reactive knowledge.
- The *Task Model* describes the task decomposition and details if a task is solved by an user or an agent
- The *Cooperation Model* specifies the communication primitives and interaction protocols required to cooperate and resolve conflicts
- The *System Model* defines the organizational aspects of the agent society together with the architectural aspects of agents

- The *Design Model* collects the previous models and captures the non functional requirements required to operationalize them.

MAS-CommonKADS [Iglesias et al., 1997] extends the models defined in CommonKADS adding techniques from object-oriented methodologies (OOSE, OMT) and from protocol engineering (MSC and SDL). This methodology starts with an informal *conceptualization phase* used to obtain the user requirements and a first description of the system from the user point of view. For this purpose, *use cases* from OOSE are used, and its interactions are formalized with Message Sequence Charts. Then, the different models described below are used for analysis and design of the system, that are developed following a risk-driven life cycle. For each model the methodology defines the constituents (entities to be modelled) and the relationships between the constituents. A textual template and a set of activities for building each model are provided according to a development state (empty, identified, described or validated). MAS-CommonKADS defines the following models:

- *Agent model*: describes the main characteristics of agents, including capabilities, skills (sensors/effectors), services, goals, etc.

- *Task model*: describes the tasks (goals) carried out by agents and tasks decomposition, using textual templates and diagrams.

- *Expertise model*: follows the KADS approach, that distinguishes domain, task, inference and problem solving knowledge. The MAS-CommonKADS methodlogy proposes a distinction between autonomous PSMs, that can be carried out by the agent itself, and cooperative PSM, that decompose a goal into subgoals that are carried out by the agent in cooperation with other agents.

- *Coordination model*: describes the conversations between agents. A first milestone is intended to identify the conversations and the interactions. The second milestone is intended to improve conversation with more flexible protocols such as negotiation and identification of groups and coalitions. The interactions are specified using MSC (Message Sequence Charts) and SDL (Specification and Description Language).

- *Organization model*: describes the organization in which the MAS is going to be introduced and the organization of the agent society. The agent society is described using an extension of the object model of OMT, and describes the agent hierarchy, the relationship between the agents and their environment, and the agent society structure.

- *Communication model*: details the human-software agent interactions, and the human factors for developing these user interfaces.

- *Design model*: collects the previous models and is subdivided into three submodels: *application design*, *architecture design*, and *platform design*. The application design is about the composition or decomposition of the

agents according to pragmatic criteria and selection of the most suitable agent architecture for each agent. The architecture design deals with the relevant aspects of the agent network: required network, knowledge and telematic facilities. The platform design refers to the selection of the agent development platform for each agent architecture.

Other approaches

Several formal approaches have tried to bridge the gap between formal theories and implementations [d'Inverno et al., 1997]. Formal agent theories are specifications that allow the complete specification of agent systems. Though formal methods are not easily scalable [Fisher et al., 1997], they are specially useful for verifying and analyzing critical applications, prototypes and complex cooperating systems. Some examples include the use of Z [Luck et al., 1997], temporal modal logics [Wooldridge, 1998], and DESIRE [Brazier et al., 1997]. DESIRE (DEsign and Specification of Interacting REasoning components) proposes a component-based approach to specify the following aspects: task decomposition, information exchange, sequencing of subtasks, subtask delegation and knowledge structures. It is well suited to specify the task, interaction and coordination of complex tasks and reasoning capabilities in agent systems.

During the development of Multi-Agent Systems some developers have adopted a software engineering approach that can be used as the basis for an agent oriented methodology. Although in some cases they have not explicitly defined an agent oriented methodology, they have given general guidance, and in some other cases, they have proposed a methodology based on their personal experience. Some examples are:

- In *ARCHON* [Wittig et al., 1994], the analysis combines a *top-down* approach, that identifies the system goals, the main tasks and their decomposition, and a *bottom-up* approach, that allows the reuse of preexisting systems, thus constraining the top-down approach. The design is subdivided into *agent community design* (defines the agent granularity and the role of each agent) and *agent design* (encodes the skills for each agent).

- *MADE* [O'Hare and Woolridge, 1992] is a development environment for rapid prototyping of MAS. It proposes a methodology that extend the five stages of knowledge acquisition proposed in [Buchanan et al., 1983]: Identification, Conceptualization, Decomposition (added for agent identification), Formalization, Implementation and Testing (adding the integration of the MAS).

- The *AWIC* [Müller, 1996] method proposes an iterative design. In every cycle five models are developed: Agent model (tasks, sensors and actuators, world knowledge and planning abilities), World model, Interoperability model (between the world and the agents), and Coordination model (protocols and messages, study the suitability of joint plans or social structuring).

- The *Decentralizing Refinement Method* [Singh et al., 1993] proposes to start with a centralized solution to the problem. Then a general PSM is abstracted out. Next step is the identification of the assumptions made on the agent's knowledge and capabilities, and the relaxation of these assumptions in order to obtain a more realistic version of the distributed system. Finally, the system is formally specified. The method takes into account the reuse of the PSMs by identifying connections among parts of the problems and the agents that solve them.

2.4.6 Conclusions

Most research in the field of Cooperative Problem Solving (CPS) falls within the context of the Cooperative Problem Solving process [Wooldridge and Jennings, 1994], with four stages: recognition, Team Formation, planning and execution. In this framework the problem solving process starts with an agent willing to solve a task and realizing the potential for cooperation, but the process of deciding the goals to achieve and the way to achieve them is skipped, assuming that they are provided by the user [Wooldridge and Jennings, 1999]. Moreover, task allocation among cooperating agents is typically based on a preplan that decomposes a task into subtasks [Shehory and Kraus, 1998], without specifying the algorithms to build such a plan, neither the criteria to be taken into account, e.g. the Planned Team Activity [Sonenberg et al., 1994] and the SharedPlans approach [Grosz and Kraus, 1996]. Our work focuses on the feasibility and utility of a componential approach to build such initial plans using the the knowledge-level description of the Multi-Agent system (i.e. building a configuration of tasks, capabilities and domain knowledge).

When addressing the problem of designing the behavior of a Multi-Agent System we agree with other researchers that users matter [Erickson, 1996b]: people may need to understand what happened and why a system alters its responses, have some control over the actions of the system, even though agents are autonomous, or predict the overall system behavior. There is a need for methods to guide the Team Formation process according to stated problem requirements and user needs. Existing frameworks for developing cooperative MAS assume that both team plans and individual plans are known beforehand, and the stage called "planning" in fact is a re-planing stage, because agents refine the initial plans until a agreed plan is decided.

We claim the need of a framework with two integrates the problem specification within the CPS process; and second, provides a fully automated configuration process (equivalent to build a plan) of a MAS on-demand. The idea of the configuration process is to build a hierarchical TMD structure encompassing the capabilities required for a Team to solve a particular problem, and this process is driven by problem requirements in place of beforehand plans. We contribute to this issue by introducing a Knowledge Configuration process as the initial stage of the Cooperative Problem Solving process. Such a TMD configuration is an extension of the idea of matchmaking to deal with the domain, which facilitates

the reuse of existing capabilities not only for new requirements, but also for new application domains.

There is another MAS topic we have to have a closer look, *matchmaking*. While existing frameworks support matchmaking between tasks and capabilities, we are also considering matchmaking among capabilities and domain models in order to enhances the reusability of agent capabilities across different application domain. Moreover, the ORCAS Knowledge Configuration process fills the gap between the matchmaking process, that pairs a specification to a capability, and the global configuration of a team plan, which is required to solve the "bottom-up design problem" (§2.2): *given a set of requirements, find a set of agent capabilities and domain knowledge within a MAS whose aggregated competence satisfies the requirements*. The fundamental difficulty when considering this problem is how to decompose the requirements in such a way as to yield component specifications (i.e. capabilities and domain-models). Our approach to this problem is to search the space of all possible compositions of components —configurations— until one satisfying the requirements is found. Moreover, our work has extended the Knowledge Configuration process to apply Case-Based Reasoning during the selection of components. The idea is that past experience can be used to improve the search process by guiding the exploration of possible configurations according to the similarity of the current problem to past configuration problems. The reader is referred to §4.4 for the general configuration strategy and §4.5 for the case-based configuration approach

Concerning the topic on Agent Capability Description Languages (ACDL), most approaches distinguish among tasks (or goals) and capabilities (or services), but these components are tightly coupled to a particular application domain. Our approach to improve the reuse of agent capabilities is using the TDM approach for describing agent capabilities and tasks in a domain independent manner, abstracted from the application domain. This decoupling of capabilities and domain is enabled by allowing agent capabilities and domain knowledge to be described independently, as proposed by TMD frameworks to specify knowledge-systems. Capability descriptions are compared to domain-models during the Knowledge Configuration process to verify that the domain knowledge satisfies the capability assumptions, and this process is fully automated to enable the on-demand, on-the-fly configuration of the MAS.

In addition to the reuse issue, there are another aspects of MAS design that can benefit from a Knowledge Modelling framework:

- Domain knowledge acquisition is facilitated by an abstract description or model of the domain knowledge required for some application, that can be used as a guide during the knowledge acquisition process.

- Problem specifications are also oriented by the abstract level description of a system, which facilitates the task of posing appropriate problem requirements to the user.

Moreover, there are some limitations of TMD frameworks to be applied in MAS configuration. These methodologies conceive a knowledge system as a cen-

tralized one, without considering neither the social dimension of agents, nor the issue of autonomy. On the other hand, agents are autonomous entities that can decide if accepting or refusing requested actions. Consequently, TMD frameworks must be extended to allow some kind of distributed control and coordination rather than applying a centralized control schema. On the other hand, usually agents interoperate through a conversational model, using speech-acts to communicate the purpose of a message, and following structured interaction protocols. In consequence, and to sum up, our framework should deal with specific agent properties, deserving special attention to the fact agents are autonomous and communication is based on speech-acts and interaction protocols.

Concerning software libraries and MAS, the idea of reusable software libraries is related to concepts from Multi-Agent Systems, like *directory facilitator* or *yellow pages* services. In addition, selection of components in software reuse is defined in terms matching, in a similar way to the *matchmaking* process as performed by middle agents in open MAS. Our approach aims at integrating research on software libraries and reuse together with recent work on open agent architectures, more specifically, we propose the following ideas:

- that agent capabilities can be registered and managed as components in a library (or a middle agent), and can be queried by other agents in order to locate them;
- and knowledge modelling can be used to describe the components with semantic information, abstracting them from implementation details in order to maximize reuse.

2.5 Semantic Web services

Semantic Web Services are defined as "self contained, self describing modular applications that can be published, located and accessed across the Web" [Tidwell, 2000], and also as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard internet protocols" [McIlraith et al., 2001]). Today's Web was designed primarily for human interpretation, to be used as a repository of information. But nowadays the Web is becoming also an open environment for distributed computing, where new applications can be built by assembling information services on-demand from a montage of networked legacy applications and information sources [International Foundation on Cooperative Information Systems, 1994].

Web services technologies are beginning to emerge as a defacto standard for integrating disparate applications and systems [Peltz, 2003]. Nevertheless, most Web service interoperation is realized through APIs that incorporate hard code to locate and extract content from HTML pages. These mechanisms are not suited to deal with an open, changing environment like the Internet. In order to implement reliable, large scale interoperation of Web services it is fundamental to make such services computer interpretable, which could be achieved by

creating a Semantic Web [Berners-Lee et al., 1999] of services whose properties, capabilities, interfaces and effects are encoded in an unambiguous, machine understandable form. The realization of the Semantic Web is underway with the development of new markup languages with well defined semantics and able to manipulate complex relations between entities [Stab et al., 2003]. Some examples of such languages are OIL [Fensel et al., 2000], DAML+OIL [Horrocks, 2002] and DAML-S [The DAML-S Consortium, 2001].

Semantically annotated Web services can support the automated discovery, execution, composition and interoperation of services by computer programs or agents [McIlraith et al., 2001]. The distributed control nature of agents make them suited to become accessible through the Internet likewise services. From that view, agent capabilities can be seen as services provided to users or other agents through mediation services such as yellow pages; therefore, developing agent-based applications by reusing existing agent capabilities in open environments like the Internet will face the same problems encountered in the research on Semantic Web Services; and there is likewise a need for languages and infrastructures supporting the automated discovery, execution, composition and interoperation of agent capabilities.

Agent description languages could be used for describing Web services and viceversa with little adaptation effort. Markup of services exploits ontologies to facilitate sharing, reuse, composition, mapping and markup [Fensel et al., 1997, Fensel and Bussler, 2002]. Service description should include a functional view to enable automatic service discovery, a pragmatic view, or some kind of operational metrics are also very desirable, e.g. QoS [Cardoso and Sheth, 2002], but also a process model of the service designed to facilitate service composition. While there are a considerable level of consensus with respect to the functional aspects of a service (inputs, outputs, preconditions or prerequisites and postconditions or effects), and few standardization proposals (WSDL, DAML-S profile ontology), the panorama is quite different when talking about the process model. There are a lot of languages proposed for this purpose, including WSFL, XLANG, WSCI, BPML, BPEL4WS and YAWL [van der Aalst and ter Hofstede, 2002]. These languages are called Web Service composition languages, workflow languages, process languages, and Web Service orchestration languages.

An introduction to the Web Services approach can be found in [Vaughan-Nichols, 2002] and recent trends and controversies in [Stab et al., 2003].

2.5.1 Semantic Web Services Frameworks

For an overview of existing technologies and research directions on service description, advertising and discovery, the reader is referred to [Lemahieu, 2001]. A brief review of the most influencing proposals is provided below:

- The Web Service Modelling Framework (WSMF) [Fensel and Bussler, 2002, Bussler et al., 2002] proposes a conceptual

model for developing and describing services and their compositions. WSMF is based on maximal decoupling and scalable mediation services.

- DAML-S [The DAML-S Consortium, 2001, Ankolekar et al., 2002] is a DAML+OIL [Horrocks, 2002] ontology for describing the properties and capabilities of Web Services. The purpose of the DAML-S ontology is to allow automatic reasoning about Web services in order to improve the discovery, access, composition and interoperation of Web services. The approach to do that is to enrich service markup with semantics. An example of semantic matching for service discovery is described in [Paolucci et al., 2002].

- DLML [Euzenat, 2001] proposes to describe Web services using a Description Logics Markup Language (DLML) for ensuring interoperability among semantically heterogeneous Web services. DL allows to formally define transformations(mappings), proof of properties, and checking of compound transformations.

- WSTL Web Service Transaction Language [Piresa et al., 2003]. Extends WSDL for enabling the composition of Web services.

- Web Services Semantic Architecture is based upon a mix of standard Web services technology (UDDI, WSFL and DAML-S) and the DAML-S ontology.

2.5.2 Composition and interoperation of Web services

The most extended approach for composing Web services is the use of a *workflow* language. But there is a lack of consensus [Wil M. P. van der Aalst, 2002], which results in a variety of languages to describe service composition from a workflow approach , like WSFL, XLANG, WSCI, BPML and more recently BPEL4WS (see for instance [Peltz, 2003] for a review of emerging technologies, tools and standards). A particular approach is to use Petri Nets as the computational formalism of a workflow language, like WRL [van der Aalst et al., 2001] and YAWL [van der Aalst and ter Hofstede, 2002]. A great challenge is to integrate heterogeneous services, which requires to address the interoperability issue not only at the syntactic level, but also at the semantic level. For example, [Cardoso and Sheth, 2002] describes an ontology-based approach for the discovery and integration of heterogeneous Web services within workflow processes. This approach takes into account both functional and operational requirements (Quality of Services) and provides some algorithms to measure the degree of integration.

Action-based planning [Wil M. P. van der Aalst, 2002] is an approach to services composition that views Web Services as a collection of actions available to build a plan. An example of this approach is demonstrated by ConGolog [McIlraith et al., 2001, McIlraith and Son, 2001], a concurrent logic programming language based on *situation calculus*. ConGolog can be used to program

agents which can perform simulation and execution of composite Web services customized for the user.

Transactional approaches are inspired by a business perspective; for example, in [Piresa et al., 2003] a multi-layered mediated architecture is presented together with a language to compose Web services from a transactional viewpoint.

To conclude, there is yet another approach that thinks of Web services as behavioral extensions of agent capabilities. From this point of view, Web services are located, invoked, composed and integrated by intelligent software agents [Bryson et al., 2002, McIlraith and Son, 2001]. DAML-S is proposed as a language for the semantic markup of Web services. Semantic markup of services could enable to apply simulation, verification and automatic composition of services, for instance using Petri Nets [Narayan and McIlraith, 2002]. An agent-oriented architectural framework for Web services based on the notion of a Flexible Agent Society (FAS) has been proposed [Narendra, 2003] based on the Contractual Agent Society (CAS).

2.5.3 Conclusions

Semantic Web Services (SWS) is an emergent field. While there is a well defined approach to describe Semantic Web Services and some generalized standards, there is still much discussion on the operational description of services, and there are many proposals of frameworks for the composition of complex services. There are some notable similarities when comparing SWS frameworks and Agent Capability Description Languages used in open MAS. There are, however, some notable differences between both fields: While SWS are passive entities that are executed by direct invocation, agent capabilities are provided by autonomous agents that can decide autonomously whether to apply or not to apply a required capability, or the terms of commitment when accepting some request. Consequently, ACDLs can benefit only partially of SWS research, specifically, ACDLs and SWS languages share may common requirements, but differ on the operational aspects of composition: while SWS are well suited for a centralized control, autonomous agents are appropriate for a distributed control style.

SWS frameworks are designed to facilitate reuse and composition of services to achieve more complex tasks within a concrete domain, but SWS are not oriented towards domain-based reuse. Typically the knowledge used by a SWS is encapsulated or hidden, thus it is not possible to reason about the domain knowledge, neither to use a service for a new domain. Configuring a MAS on-demand from reusable capabilities and knowledge, requires a language for describing and reasoning about a MAS at an abstract level. But, although this level could be provided by semantic languages and compositional frameworks based in either SWS or TMD frameworks, there is required much work to integrate these approaches with cooperative MAS.

Chapter 3

Overview of the ORCAS framework

This chapter provides an overall view of the ORCAS framework that accounts for its multi-layered structure, and highlights the outstanding points.

Now that we have reviewed the most relevant bibliography, it is time to summarize which are the open problems we are dealing with and the kind of solutions we propose; to get a view of the tree in order to avoid getting lost when accounting for the leaves.

The main goal of this thesis is to provide a framework for open Multi-Agent Systems that maximizes the reuse of agent capabilities through multiple application domains, and supports the automatic, on-demand configuration of agent teams according to stated problem requirements.

There are some agent infrastructures relying on formal languages for describing both available capabilities and requests to solve problems using those capabilities; these infrastructures are usually based on middle agents to handle the interoperation issues. Middle agents — matchmakers and brokers— are able to match requests to advertised capabilities in order to find appropriate capability providers. However, no mechanism has been proposed to design the competence of a team in such a way that global problem requirements are satisfied. Therefore, automated design mechanisms supporting the on-demand, configuration of agent teams are required beyond existing matchmaking algorithms. Our proposal is to introduce a Knowledge Modelling Framework (KMF) to describe agent capabilities at a domain independent level, and to apply a compositional approach to design Multi-Agent Systems on-demand. The ORCAS KMF is based on the Task-Method-Domain paradigm, which distinguishes three classes of components: *Tasks*, *Problem-Solving Methods* and *domain-models*. In the ORCAS KMF we consider also three classes of components, namely agent *capabilities* (corresponding to Problem-Solving Methods), application *tasks* and *domain-models*.

The idea of the ORCAS KMF is to apply an abstract compositional architecture to configure agent-based applications on-the-fly, by selecting and connecting components that satisfy the requirements of each specific problem. To do so, we have transferred the "bottom-up design problem" [Mili et al., 1995] from the field of software reuse (§2.3) to the field of Multi-Agent Systems. As a result, we have defined the problem of designing a team as follows:

> given a set of requirements, find a set of agent capabilities and domain-models whose combined competence satisfy the requirements.

The main difficulty when considering a 'bottom-up design problem" is how to decompose the requirements in such a way as to yield component specifications. We adopt here an abstract view that approaches this problem as a search process, namely one that can be solved by a search process over the space of all possible component compositions, until one satisfying the requirements is found [Hall, 1993, Zhang, 2000]. Specifically, the composition of components is regarded as a composition of their specifications [Butler and Duke, 1998], which in our case are provided by the Knowledge Modelling Framework and consist of tasks, capabilities and domain-models.

Our proposal to configure a MAS is the separation of two layers in the configuration process: the knowledge and the operational layers. At the knowledge layer a MAS is described and configured in terms of component specifications and connections, at an abstract, implementation independent level. At the operational layer a team of agents is formed and each agent receives instructions on how to cooperate and coordinate in solving a problem according to this abstract (knowledge-level) configuration. Figure 3.1 shows the two layers MAS configuration model.

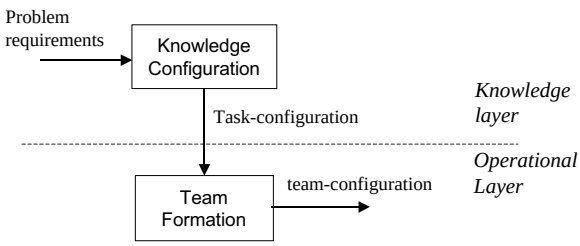


Figure 3.1: The two layers MAS configuration model.

(1) Configuration at the *knowledge layer* refers to the process of finding a configuration of components (tasks, capabilities and domain-models) adequate for the problem to be solved. We call this process *Knowledge Configuration*, and the result of the process is a *task-configuration*: a hierarchical structure of tasks, capabilities and domain-models satisfying the problem requirements. The Knowledge Configuration process is intended to take a specification of problem

requirements as input and producing a configuration of components such that the problem requirements are satisfied. The main goal of the Knowledge Configuration process is to determine which task decomposition, which competencies and which domain knowledge is required by a team of agents to solve a given problem.

(2) Configuration at the *operational layer* refers to the process of operationalizing a configuration into an executable system; in other words, to form a team of agents that is equipped with the capabilities and knowledge specified by a task-configuration (it means that the team is customized for the problem at hand). The operational configuration layer has the goal of ensuring that the multiple components involved in a task-configuration can interoperate and cooperate to solve a problem together.

There are some efforts on using a componential approach for the compositional-specification and design of intelligent agents [Decker et al., 1997a, Herlea et al., 1999, Splunter et al., 2003], and Multi-Agent Systems [Brazier et al., 2002], but these frameworks are mostly oriented to the development of agents, whereas our work is about forming and customizing agent teams on-demand, by composing components provided by already existing agents. We take the view on Multi-Agent Systems as decoupled networks of autonomous entities associated to a distributed model of expertise, regarded as a collection of specialized agents with complementary skills. According to this setting, a general framework for the Cooperative Problem Solving (CPS) process has been described [Wooldridge and Jennings, 1999] as having four stages: recognition (an agent identifies the potential for cooperation), team formation (the process of selecting team members), plan formation (collective attempts to construct an agreed plan) and execution (agents engage in cooperative efforts to solve the problem according to the agreed plan).

The result of separating the configuration of a MAS in two layers is a new model of Cooperative Problem Solving process that includes a Knowledge Configuration process before the Team Formation process. Moreover, the planning stage as presented in existing frameworks is removed from the ORCAS framework, since planning is usually associated to a particular agent architecture and is focused on internal agent processes, while our approach here is to take a macro-view, focused on the external, observable phenomena of teamwork, rather than imposing a particular agent model. In some aspects, the role played by the planning stage in the CPS process is substituted by the Knowledge Configuration process, since a task-configuration is a kind of global plan that will be used as a recipe to guide the team formation and to coordinate agents during the execution stage, that we call the Teamwork process (notice that the CPS model includes the Teamwork process within).

The ORCAS model of the Cooperative Problem Solving process comprehends four sub-processes, as showed in Figure 3.2, namely Problem Specification, Knowledge Configuration, Team Formation and Teamwork. The result of the Problem Specification process is a specification of a set of problem requirements to be satisfied, and problem data to be used during the Teamwork process.

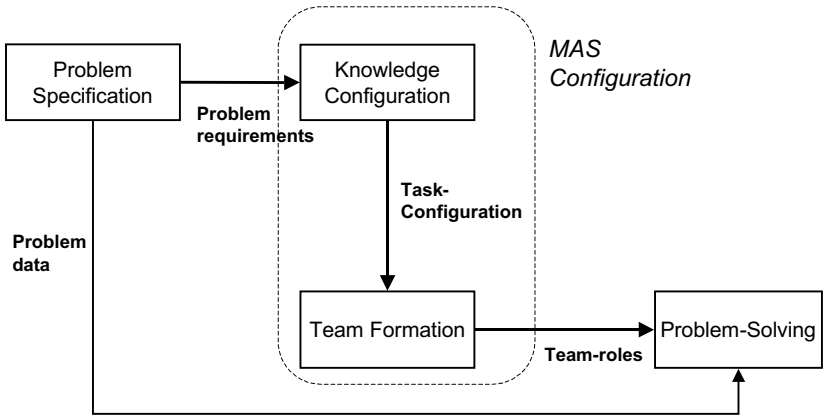


Figure 3.2: Overview of the ORCAS Cooperative Problem Solving process

The Knowledge Configuration uses the problem requirements to produce a task-configuration, which is used to guide the Team Formation process. Next, during the Teamwork process, the configured team resulting of the Team Formation stage applies the capabilities and knowledge specified in the task-configuration in order to solve the problem.

We claim that the separation between the knowledge (abstract) layer and the operational (computational) layer helps to distinguish between the static and dynamic aspects of agents to be taken into account during the configuration of a MAS. The idea is to exploit the fact that the abstract specification of agent capabilities remains stable over long periods of time, whereas there are dynamic aspects of the system or its environment that change very quickly, e.g. the agent workload or the network traffic. Therefore, it is useful to make a task-configuration in terms of a stable, abstract description of capabilities, and thereafter use the task-configuration to select the "best" candidate agents[1] according to dynamic and context-based information. While other frameworks and infrastructures focus on the task allocation stage carried on during Team Formation, the Knowledge Configuration process is situated just before Team Formation in the ORCAS Cooperative Problem-Solving model.

However, the CPS model should not be understood as a fixed sequence of steps, it is rather a process model of CPS. Thus, although the Knowledge Configuration is situated before Team Formation in the model, this does not imply that the Knowledge Configuration process should be fully completed before Team Formation begins. In fact, we have implemented strategies that interleave both activities with Teamwork, enabling distributed configuration, lazy configuration and dynamic reconfiguration on runtime (§5.7).

[1]The notion of agent goodness is specified as a criteria to be optimized, i.e. cost, speed, reliability, etc. and the possible trade-offs among them.

An interesting issue addressed by our framework concerns Team Formation in large systems: agent selection during Team Formation may involve an exponential number of possible team combinations, and a blow-out in the number of interactions required to select the members of a team. The performance of the Knowledge Configuration process brings about a task-configuration that can be used to guide the Team Formation process: only agents with capabilities selected during the Knowledge Configuration process are candidates to join a team, thus reducing the number of potential teams to be considered during the selection of team members, and drastically decreasing the amount of interaction required among agents. Throughout, the combinatorial problem is transferred (although mitigated) from the Team Formation process to the Knowledge Configuration process. Our proposal to further improve this issue is the use of Case-Based Reasoning to constrain the search over the space of possible configurations. The idea is to use past configuration problems (cases) to heuristically guide the search process over the space of possible configurations, as explained in §4.5.

Lastly, our objective is to develop an open agent infrastructure backing the on-demand configuration of Cooperative MAS according to stated problem requirements, based on both the Knowledge Modelling and the Operational Framework. Our purpose is to provide an open but trustworthy infrastructure where to test the proposed frameworks. Specifically, we propose to introduce a social mediation layer where specialized agents provide the services required to perform the different processes of the CPS model: Problem Specification, Knowledge Configuration, Team Formation and Teamwork. Since we want to avoid imposing architectural constrains over individual agents, we adopt a macro-view centered on the interaction protocols and communication language rather focusing on any particular agent architecture. Therefore, we decided to use a social oriented approach to build an open agent infrastructure, and more specifically, we adopted the *electronic institutions* (§2.4.4) approach.

An electronic institution is an infrastructure providing the mediation services required for agents to successfully interact in open environments under controlled conditions. This means that the institution imposes some constrains over the agents observable behavior. In other words, an electronic institution provides the specification of the *rules of encounter* for a successful interaction. The theoretical loss of autonomy that supposes joining an electronic institution brings, however, the advantage of allowing external agents to be more informed about other agents, since the overall system behavior becomes more predictable by virtue of the institution. The ORCAS infrastructure has been designed and implemented as an electronic institution, that we call the ORCAS e-Institution. In the ORCAS e-Institution requesters and providers of capabilities join and interact by using the services provided by institutional agents. Institutional agents are middle agents offering services beyond the usual matchmaking service. In particular, the ORCAS e-Institution includes agents that are able of: (1) keeping a repository (a library) of components available in a MAS, including application tasks, agent capabilities, and domain-models; (2) configuring the task to be achieved by a team in order to solve a given problem, according to an ab-

stract specification of problem requirements; (3) forming and instructing agent teams that are customized for each particular problem; and (4) coordinating team members behavior during the Teamwork process.

The main outcome of this work is a two-layered framework for integrating Knowledge-Modelling and Cooperative Multi-Agent Systems together, called ORCAS that stands for Open, Reusable and Configurable multi-Agent Systems. The feasibility of the ORCAS framework has been demonstrated by implementing the ORCAS e-Institution, an infrastructure for MAS development and deployment that supports the on-demand configuration of teams and the coordination of agent behaviors during teamwork, according to the ORCAS two-layered framework. The applicability of the framework has been tested by building WIM a multi-agent application running upon the ORCAS e-Institution. WIM is a configurable MAS application to look for bibliographic references in a medical domain, and is explained in Chapter 7. We show that the clear separation of layers will support a flexible utilization and extension of the framework to fit different needs, and to build other infrastructures different from the implemented ORCAS e-Institution.

These are the two layers of the ORCAS framework, namely the Knowledge Modelling Framework, and the Operational Framework:

1. The *Knowledge Modelling Framework* (KMF) (Chapter 4) is about the conceptual description of a problem-solving system from a knowledge-level view, abstracting the specification of components from implementation details. Our approach is to describe agent capabilities abstracted from implementation details in order to support the configuration of the MAS independently of the programming language and the agent platform. The configuration of a Multi-Agent System in terms of abstract (knowledge-level) descriptions is automated to enable the MAS configuration to occur on-demand, according to the requirements of the problem at hand. This layer consists of an Abstract Architecture, and Object Language and a Knowledge Configuration process:

    - The *Abstract Architecture* defines the different components used to model a MAS, which is based on the Task-Method-Domain-modelling paradigm; the features proposed to describe each component; and the functional relations that constrain the way components can be connected to become a meaningful system. The elements of the Abstract Architecture are explicitly declared as an ontology, called the Knowledge Modelling Ontology.
    - The *Object Language* is the language used to formally specify component features, and the inference mechanism used to reason about the specification of components (e.g. to determine if a capability is suitable to solve a task).
    - The *Knowledge Configuration* process is a search process aiming at finding a configuration of components (tasks, capabilities and domain-models) fulfilling the specification of the problem at hand. The result

of the Knowledge Configuration process is a hierarchical decomposition of the initial task into subtasks, capabilities bound to each task, and domain models bound to capabilities requiring domain knowledge. The result of the Knowledge Configuration process is called a *task-configuration*.

2. The *Operational Framework* (Chapter 5) describes the link between the characterization of the problem solving components and its implementation by Multi-Agent Systems. This framework describes how a task-configuration at the knowledge-level can be operationalized into a team of agents that is formed on-demand and is customized to satisfy the specific requirements of each problem. This layer introduces a *team model* based on the KMF, the ORCAS ACDL, the Team Formation process, and the Teamwork process.

   - The *Team model* describes the structure and the organization of teams according to a task-configuration as obtained by the Knowledge Configuration process. The team model establishes a mapping between KMF concepts and concepts from teamwork and Multi-Agent Systems.
   - The *Agent Capability Description Language* (ACDL) is a language for describing and reasoning about agent capabilities in such a way that enables the automatic location (by performing matchmaking), invocation, composition, and monitoring of agent capabilities. The ORCAS ACDL is defined as a refinement or extension of the Knowledge Modelling Ontology. Specifically, the ORCAS ACDL introduces an agent based formalism to describe the operational aspects of a capability: the *operational description* and the *communication*.
   - The *Team Formation* process deals with the selection of agents to join a team, and the instruction of the selected team members to solve a specified problem according to a task-configuration, as established by the Knowledge Configuration process. During the Team Formation process, the tasks required to solve a problem are allocated to suitable agents through a bidding mechanism, and selected agents receive instructions on which roles to play and how to cooperate with the rest of the team. The result is a collection of team roles and commitments of each team-member to their assigned roles that allows the team solving the overall problem.
   - The *Teamwork* process addresses the interaction and the coordination required for teams to successfully solve a problem according to requirements of a task-configuration. During the Teamwork process, the members of a team suitable for the problem (as established by the Team Formation process) engage in cooperative work until the global problem is solved, a reconfiguration of the team is needed, or it is impossible to solve the problem. Cooperation is basically a process of delegating subtasks to other agents in a team and aggregating

or distributing the results of the different subtasks at appropriate synchronization points.

	Architecture	*Language*	*Processes*
Knowledge Layer	Abstract Architecture	Knowledge-Modelling Ontology and Object Language	Problem Specification & Knowledge Configuration
Operational Layer	Team model	ACDL (Communication and Coordination)	Team Formation & Teamwork

Table 3.1: Summary of the two-layered framework

Table 3 is summarizes the main elements addressed at each layer, attending to three aspects: architectures, languages and processes. Furthermore, we can consider the ORCAS e-Institution as a third layer referring to a particular implementation of the former layers. We can then represent the two ORCAS layers plus the ORCAS e-Institution as a pyramid, as illustrated in Figure 3.3. The layer at the bottom addresses the more abstract issues, while upper layers correspond to increasingly implementation dependent layers. Therefore, developers and system engineers can decide to use only a portion of the framework, starting from the bottom, and modifying or changing the other layers according to its preferences and needs.

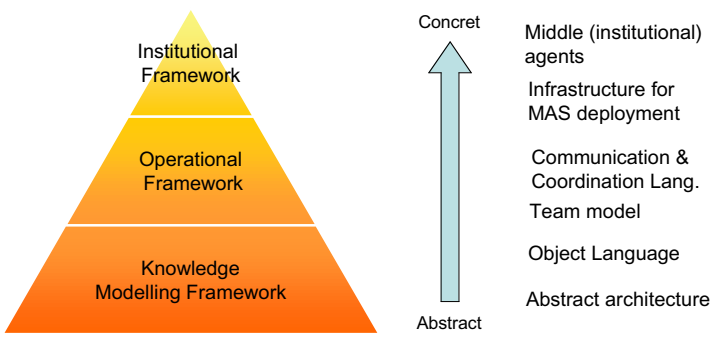


Figure 3.3: The three layers of the ORCAS framework

The structure of the thesis follows this direction from the most abstract layer to the more concrete, implementation dependent layers. Chapter 4 describes the KMF; Chapter 5 deals with the Operational Framework; Chapter 6 describes the implemented ORCAS e-Institution, and Chapter 7 demonstrates the feasibility of the framework through examples of an implemented application (WIM).

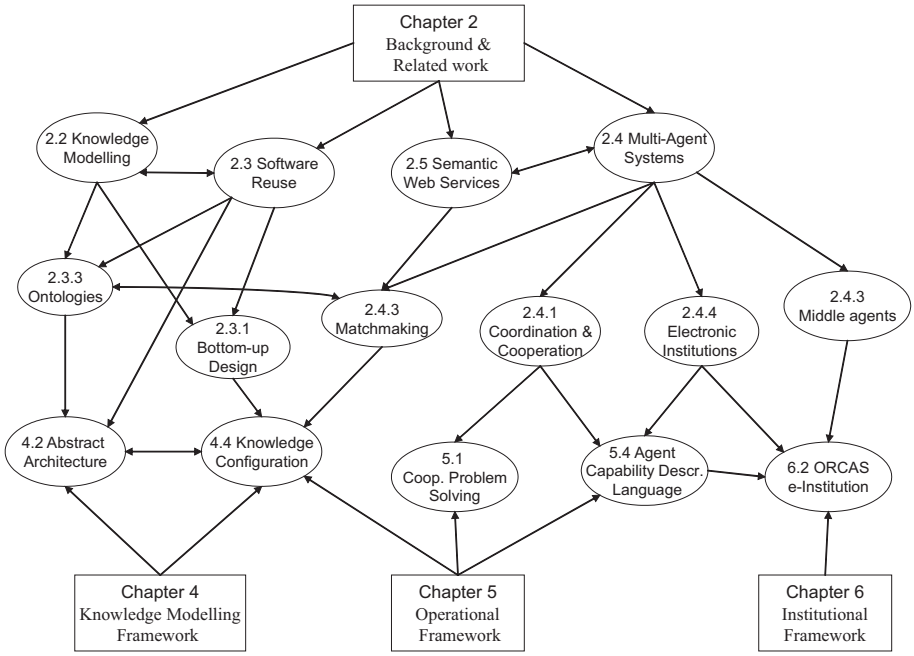


Figure 3.4: Cognitive map for the main topics involved in ORCAS

Figure 3.4 shows a map of the main topics addressed within the ORCAS framework and the relation with the subjects reviewed in the related work (Chapter 2).

Chapter 4

The Knowledge Modelling Framework

This chapter describes a framework to specify agent capabilities at the knowledge-level, which allows to reason about agent capabilities in order to design the competence of agent teams according to the requirements of the problem at hand.

4.1 Introduction

The *Knowledge Modelling Framework* (KMF) is a framework for describing and configuring Multi-Agent Systems from an abstract, implementation independent level.

The purpose of the KMF is twofold: On the one hand, the KMF is a conceptual tool to guide developers in the analysis and design of Multi-Agent Systems in a way that maximizes reuse; on the other hand, the KMF provides the basis of an Agent Capability Description Language supporting the automatic configuration of Multi-Agent Systems according to stated problem requirements.

The Knowledge Modelling Framework (KMF) proposes a conceptual description of cooperative Multi-Agent Systems at the *knowledge level*[Newell, 1982], abstracting the specification of components from implementation details. This framework is designed to maximize reuse and to support the formation and the coordination of customized agent teams during the *Cooperative Problem-Solving* process [Wooldridge and Jennings, 1999].

The KMF is the more abstract layer of the ORCAS framework for cooperative MAS. This layer consists of three main elements, namely: the *Abstract Architecture*, the *Object Language*, and the *Knowledge Configuration* process:

- The *Abstract Architecture* defines the types of components in the model, the features required to describe each component, and the relations constraining the way in which components can be connected. The ORCAS

Abstract Architecture is based on the Task-Method-Domain paradigm prevailing in existing Knowledge Modelling frameworks, which distinguish between three classes of components: *Tasks*, *Problem-Solving Methods* and *Domain-models*. In ORCAS there are tasks and domain models, while PSMs are replaced by agent capabilities, playing the same role than a PSM, but including agent specific features, like a description of a communication protocol and other features required of an Agent Capability Description Language. Nonetheless, in order to keep the KMF independent of agent details, these agent-specific aspects of ORCAS components remain unspecified here, and are described at the Operational Framework (Chapter 5).

- The *Object Language* defines the representation language used to formally specify component features. Many languages can be used as the Object Language, as far as they provide a way of specifying component features as *signatures* and *formulae*, and endorsing an inference mechanism enabling automated reasoning processes over component specifications.

- The *Knowledge Configuration* process is a search process aiming at finding a configuration of components (tasks, capabilities and domain-models) fulfilling the specification of the problem at hand. The result of the Knowledge Configuration process is a hierarchical decomposition of the initial task into subtasks called a *task-configuration*.

This chapter is organized as follows: Section §4.2 describes the components of the architecture and the relations between components that determines if two components can be connected (matching relations); Section §4.3 justifies the approach adopted here to the Object Language as a way to increase the flexibility of the ORCAS KMF, and introduces a specific Object Language; next, Section §4.4 describes the Knowledge Configuration process as a search process over the space of possible configurations; the specific approach for the Knowledge Configuration process using Case-Based Reasoning to guide the search process is described in §4.5; and finally, we end the chapter with a brief discussion on Knowledge Configuration reuse in §4.6.

4.2 The Abstract Architecture

The *Abstract Architecture* is a general modelling framework that is not specifically designed to describe Multi-Agent Systems, but to describe problem-solving (knowledge-based) systems in general. It rather plays the role of a skeleton that should be specialized or refined to deal with a concrete kind of software system, like Cooperative Multi-Agent Systems (CooMAS).

The main goal of the Abstract Architecture is to provide a way of specifying systems that maximizes the reuse of existing components and favors a compositional approach to software development. The Abstract Architecture specifies which are the components used to build an application (the "building blocks"),

and the way in which these components should be connected (the componential framework) in order to produce a valid application.

This architecture is intended to become a conceptual tool for the solution of the "bottom-up design problem" [Mili et al., 1995] and its application to the field of Multi-Agent Systems, stated as as one of main goals of this thesis (§1.1, bibliographic references in §2.3.1):

> given a set of requirements, find a set of agent capabilities and domain-models whose combined competence satisfy the requirements.

Each component in the Abstract Architecture is characterized by some features (i.e. inputs and outputs), but the particular language used to specify these features is independent of the Abstract Architecture, and belongs to the *Object Language*. The different components in the Abstract Architecture and the features characterizing them have been conceptualized and represented explicitly as an ontology, called the Knowledge Modelling Ontology (KMO). The KMO is used for analyzing, designing and describing problem-solving systems at an abstract, implementation independent level, thus it can be further refined to deal with specific architectures, including non agent-based architectures. In addition, this architecture is designed to facilitate the integration of heterogeneous systems whenever they share the same abstract architecture, i.e. using the same KMO for describing the components at the abstract level.

The Abstract Architecture enables a compositional approach supporting the on-the-fly configuration of Multi-Agent Systems at the knowledge level. A configuration is constructed by reasoning about the knowledge-level description of agent capabilities, tasks (goals) to achieve, and domain-models describing specific domain knowledge. From this approach, a system is described and configured by reusing and composing existing components. In particular, our view of components is based on the Task-Method-Domain (TMD) paradigm in Knowledge Modelling. TMD models propose three classes of components to model a knowledge system: *tasks*, *Problem-Solving Methods*(PSMs) and *domain-models*.

1. *Tasks* are used to characterize generic and reusable types of problems. This characterization is based on properties of the input, output, and nature of the operations that map the input to the output. Usually, there is a main task that describes an application problem, but tasks can be decomposed into subtasks with input/output relations between them, resulting in a task structure [Chandrasekaran et al., 1992].

2. *Problem-Solving Methods* (PSMs) specify the reasoning part of a knowledge system [Fensel et al., 1999]. PSMs are used to describe different ways of solving a task. A problem-solving method may decompose a task into subtasks or may apply a primitive inference step without further decomposing the task [Poek and Gappa, 1993]. PSMs can use domain knowledge to apply a reasoning step, create or change intermediary knowledge structures, perform actions to gather more data, etc. [Steels, 1990]. Problem-Solving Methods are described with independence of the domain knowledge

in order to maximize reuse [McDermott, 1988]. Some examples of problem solving methods are *hill climbing*, *cover and differentiate*, *propose and revise*, etc. [Breuker, 1994]

3. *Domain-models* describe domain specific knowledge that is used by Problem-Solving Methods to perform inference steps [Fensel et al., 1999]. Models can be constructed from different perspectives (for example, there are *functional models*, *causal models*, *behavioral models* and *structural models*) and represented in heterogeneous forms, like rules, hierarchies or networks [Steels, 1990].

4. *Ontologies* provide the terminology and its properties used to define tasks, problem solving methods and domain definitions [Fensel et al., 1999]. An ontology provides an explicit specification of a conceptualization, which can be shared by multiple reasoning components communicating during a problem solving process.

In our framework, tasks are used to describe the types of problems that a Multi-Agent System is able to solve. On the other hand, problem solving methods are used to specify the different capabilities agents are equipped with to solve tasks, wether they solve some task directly by applying some domain knowledge or by decomposing a task into subtasks and delegating them to other agents. While tasks are just generic problem specifications abstracted from any particular implementation, agent capabilities refer to concrete, implemented methods to solve problems that are provided by specific agents. Finally, domain-models are used to represent specific knowledge and information sources, whether the knowledge is provided by a shared repository, hold by an agent, or provided by an external information source.

The Abstract Architecture specifies the components and the way of connecting components that is necessary for the knowledge-level configuration of Multi-Agent Systems. Two kind of connections are included in our framework: on the one hand, connections between tasks and capabilities; on the other hand, connections between domain-models and capabilities. Such connections are based on the idea of *matching* and *mapping*. A matching relationship is a binary relation between two components that is verified by comparing its specifications. The verification of a matching relationship is often called *matchmaking* (e.g. matchmaking can be used to determine if two components are substitutable). On the other hand, a mapping function is an isomorphism between two specifications, in other words, a mapping is an explicit specification of the transformations required to match elements expressed in different terms —using different ontologies, with the same meaning (equivalent semantics).

In the ORCAS KMF the components —tasks, capabilities and domain-models— are described with explicit, independent ontologies [Fensel et al., 1997]. Because of this conceptual decoupling, ontology mappings may be required to match capabilities to tasks, and domain-models to capabilities when there is an ontology mismatch. Nevertheless, we will focus on the matching relations, assuming that the necessary ontology mappings are already

built, or assuming that all the components share the same ontologies. This is a reasonable assumption, since it is feasible and convenient to built the mappings beforehand, previously to make a component available for its use.

4.2.1 Components

There are three types of *knowledge components* in the Abstract Architecture (Figure 4.1), namely *tasks*, *capabilities* (playing the role of PSMs) and *domain-models*. Furthermore, each of these components is described using concepts defined at an explicit ontology, as indicated by the arrows in the figure. Herein every component can be specified using its own ontology, which entails that component specifications can be decoupled in order to maximize reuse (§4.2.1).

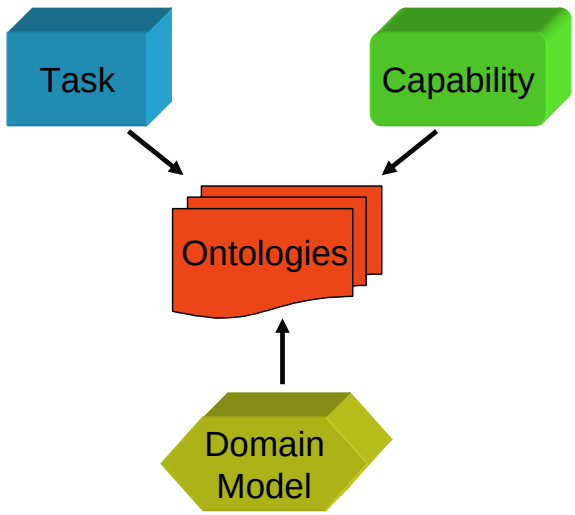


Figure 4.1: Components in the Abstract Architecture

In addition to describe component features using ontologies, the concepts used by the Knowledge Modelling Framework have been explicitly declared as an ontology, the so called Knowledge Modelling Ontology (KMO). Since the KMO is about components that are further specialized with specific component's ontologies, it can be seen a *meta-ontology* for describing software systems. Our approach is to keep a clear separation between the Abstract Architecture and the Object Language. Whilst the Abstract Architecture defines the components of the architecture and the features characterizing each component, the Object Language is used to describe component features in terms of *signature elements* and *formulae* in the Object Language.

These concepts in the KMO are organized into a hierarchy of sorts (Figure

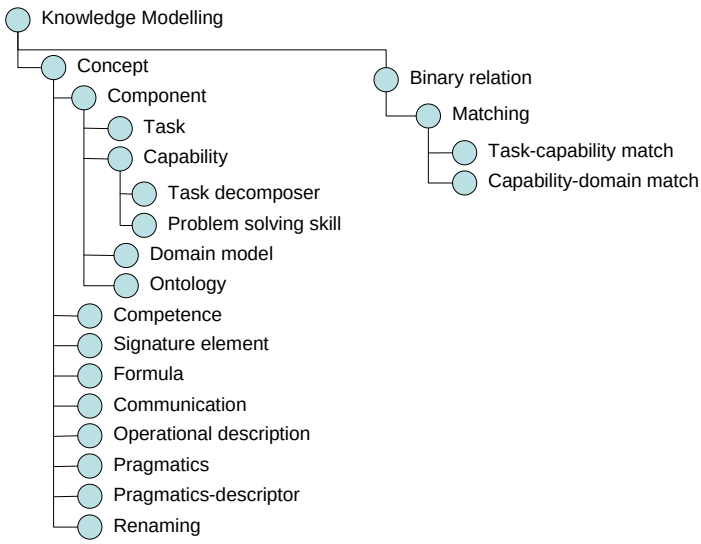


Figure 4.2: Hierarchy of sorts in the The Knowledge Modelling Ontology

4.2). There are two main sorts, namely *Component* and *Binary-Relation*, from which all the other sorts are specializations of. Most of these sorts have features that describe them, which are described by primitive types (e.g. string) or by other sorts in KMO (e.g. inputs and outputs are described with elements of the sort *Signature*).

Although the Knowledge-Modelling Framework is not dependent of any particular Object Language, the Knowledge Modelling Ontology declares two concepts that should be provided by the Object Language: *signature-elements* and *formulae*. Theses concepts are defined by the sorts *Signature-element* and *Formula* in the KMO, which should be further refined to yield a precise, interpretable meaning. In our search of a trade-off between expressive power and computational efficiency we are using *Feature Terms* as the Object Language, and subsumption as the inference mechanism (explained later, in §4.3.1).

All the components of the Abstract Architecture are subsorts of the sort Knowledge-Component. The description of elements of the sort Knowledge-Component contains a specification of pragmatics aspects of a component (e.g. name, description, creator, publisher, evaluation, etc.); and a collection of ontologies providing the terminology (the "universe of discourse") used to specify other features of a component. Moreover, the pragmatics slot can be customized by including new attributes (e.g. cost, performance measures, classification indexes, reputation, etc.) to better fit the needs or preferences of the developer.

Figure 4.3 shows the features characterizing the sort Knowledge-Component. The symbol **is-a** means subtype (subsort): *S* **is-a** *S'* means that the sort S' is a subtype (subsort) of the sort S'. The symbol → indicates the sort used to specify

Knowledge-Component **is-a** *Concept*
pragmatics → set-of *Pragmatics*
ontologies → set-of *Ontology*

Figure 4.3: The Knowledge-Component sort

a feature: f → S means that the feature f is specified as an element of sort S (an instance of S). The slot ontologies is defined as set of elements of the sort Ontology, and the pragmatics slot is described with an element of the sort *Pragmatics*. The sort *Pragmatics*, subsort of Concept (Figure 4.4) contains a number of specific features (name, creator, subject, description, publisher, etc.) specified by a String, and application descriptors, defined as a set of elements of the sort *Pragmatics-descriptor*, which are represented as attribute-value pairs (Figure 4.4). Thus, pragmatic-descriptors allow the definition and use of application specific attributes, which should be represented as attribute-value pairs.

Pragmatics **is-a** *Concept*
name → *String*
creator → *String*
subject → *String*
description → *String*
publisher → *String*
other contributor → *String*
date → *String*
format → *String*
resource identifier → *String*
source → *String*
language → *String*
relation → *String*
rights management → *String*
las date of modification → *String*
when and where be used → *String*
evaluation → *String*
application descriptors → $Pragmatics - descriptor$

Pragmatics-descriptor **is-a** *Concept*
attribute → *String*
value → *Any*

Figure 4.4: Sorts Pragmatics and Pragmatics-descriptor

Task

A *task* is a knowledge-level description of a type of problem to be solved. A task can also be seen as a set of goals characterizing "what" is to be achieve by solving some type of problem, in contrast to "how" that type of problem can be solved, which is represented as a capability. Since it is possible to have different ways of solving a problem, different capabilities might be able to solve a particular task.

The sort Task (Figure 4.5) is defined as a subsort of the sort Knowledge-Component. In addition to have a name, a description, pragmatics and ontologies, like any component, a task is described by a functional description, in terms of inputs, outputs and competence.

- The *inputs* feature represent the data required to solve the kind of problems represented by a task. This feature is specified by a set of elements of the sort *Signature* (see Figure 4.6), which is specified by a name, represented by a String; and a signature-element, of sort *Signature-element*). The sort *Signature-element* is kept undefined in the KMO, since it is refined by the sort Signature-element in the Object Language. The inputs can be constrained by preconditions in the competence.

- The *outputs* represent the type of data that is expected as a solution to the kind of problem a task represents. The *outputs*, like the inputs, are specified by a set of elements of the sort *Signature*. The outputs can be constrained by postconditions in the competence.

- The *competence* feature of a task expresses the relation between the input and the output. Since it is unfeasible to specify an input-output relation extensively (through an extensive list of input-output pairs) some form of formal representation is required. In particular, we adopt the approach of specifying the competence as a set of *preconditions* and *postconditions* (figure 4.7). Preconditions are conditions that have to hold prior to solve a task, in order to enable it, while postconditions are the properties that have to hold after solving the task (also referred in the literature as the goals or the effects to bring about). Both preconditions and postconditions are represented as a set of elements of the sort *Formula*, which is refined by the sort Formula in the Object Language. By keeping the Formula and Signature-element sorts undefined in the KMO, the Object Language could vary without affecting the Abstract Architecture, neither the matching relations at the object level (the two levels of the matching relations are explained later, in §4.2.2).

Capability

A *capability* describes the reasoning steps required to solve a class of problems (a task), that is to say, it describes the process applied by the capability to the input in order to obtain the output of the problem, and the use of required

Task **is-a** *Knowledge-Component*
pragmatics → set-of *Pragmatics*
ontologies → set-of *Ontology*
inputs → set-of *Signature*
outputs → set-of *Signature*
competence → *Competence*

Figure 4.5: The Task sort

Signature **is-a** *Concept*
name → *String*
signature-element → *Signature-element*

Figure 4.6: The Signature sort, where the Signature-element sort is to be defined by the Object Langauge

Competence **is-a** *Concept*
preconditions → set-of *Formula*
postconditions → set-of *Formula*

Figure 4.7: The Competence sort

knowledge to solve it. There are two types of capabilities, *task-decomposers* and *skills*. Skills are used to describe primitive or atomic reasoning steps, that are not further decomposed, while task decomposers are used to describe complex reasoning processes that decompose a problem into more specialized subtasks.

As showed in Figure 4.8, the sort Capability is defined as a subsort of the sort Knowledge-Component, and as such, it has two features for defining the ontologies used by the capability and its pragmatics. In addition, a capability includes a functional description that is specified as a collection of *inputs*, *outputs* and the *competence*, which is an intensional description of the capability input/output relation. Another aspect of a capability is relative to the domain knowledge it requires to operate: in order to be domain independent a capability explicitly declares the type of knowledge it can operate with. The type of domain knowledge required by a capability is specified as a collection of *signatures* in the *knowledge-roles* feature. Moreover, a capability includes a specification of *assumptions*, which are properties that should be verified by a domain-model providing some knowledge-role, in order to be sensibly used by the capability. Furthermore, a capability includes another feature called *communication*, which is used to describe the technical aspects required to invoke and interact with a capability. Since the information provided by the communication slot depends on implementation details it is explained later, within the Operational Framework (Chapter 5).

Capability **is-a** *Knowledge-Component*
pragmatics → *Pragmatics*
ontologies → set-of *Ontology*
inputs → set-of *Signature*
outputs → set-of *Signature*
competence → *Competence*
knowledge-roles → set-of *Signature*
assumptions → set-of *Assumptions*
communication → *Communication*

Figure 4.8: The Capability sort

A more detailed description of the different features characterizing a capability (Figure 4.8) follows:

- The *inputs* feature represents the data required to apply the capability. This feature is represented by a set of elements of the sort *Signature*, which is defined by sort consisting of a name and a signature element. A signature element is defined by an element of the sort *Signature-element*, to be refined by a sort in the Object Language (Figure 4.6). *Inputs* can be constrained by the the preconditions specified in the competence feature.
- The *outputs* feature represents the type of the data that is expected as a result of applying the capability to solve the tasks it is suitable for.

The *outputs* are represented by elements of the sort *Signature* and can be constrained by postconditions in the competence feature.

- The *competence* of a capability represents the relation between the input and the output, and (as for tasks) is specified as a set of preconditions and postconditions (Figure 4.7). Preconditions are conditions that have to verify in order to enable the inference process provided by the capability, while postconditions are the new conditions produced by the application of the capability (also referred as the effects brought about), whenever the preconditions hold. Both preconditions and postconditions are specified as elements of the sort *Formula*, which is defined in the Object Language.

- The *knowledge-roles* are specifications of "inputs" to be provided by some domain knowledge. Knowledge-roles refer to concepts characterizing the application domain, and are used during the Knowledge Configuration to select domain models that are compatible with a capability. Only the domain models providing the concepts required by a knowledge-role are suitable for a capability. A knowledge-role is defined as an element of the sort *Signature*, like the inputs and outputs, and as such it is defined by the sort *Signature-element*, to be refined in the Object Language.

- The *assumptions* of a capability are necessary criteria for the achievement of the desired competence of the capability. Assumptions are conditions required from a domain model providing a particular knowledge-role to be sensibly used by the capability. Assumptions are associated to a particular knowledge-role, from those specified in the *knowledge-roles* feature of the capability. Assumptions are represented as elements of the homonym sort: *Assumptions* (Figure 4.9). An assumption is specified as a pair consisting of a knowledge-role, and a collection of conditions over the domain-model providing such knowledge-role. If a capability introduces more than one knowledge-role, then a domain-model is required to fill in every knowledge-role, and each domain-model has to verify the conditions associated to that knowledge-role.

- The *communication* slot defines technical information about the interaction protocol and the data format used to communicate with the provider of the capability. Since capabilities in ORCAS are provided by agents, the communication information is basically defined by the agent communication language and some kind of interaction protocol describing the pattern of communication between the requester and the provider of the capability. The information provided by the communication slot is required during the Cooperative Problem-Solving process to request an agent to apply a capability for solving a task. Since the communication property is closer than other aspects of a capability to the implementation details, it is avoided at the Knowledge Configuration process, and is instead presented as an operational property required for the ORCAS KMF to become

a full-fledged Agent Capability Description Language. Consequently, communication aspects are described in the Operational Framework (Chapter §5).

Assumptions
knowledge-role → *Signature*
conditions → set-of *Formula*

Figure 4.9: The Assumptions sort

Skills

A *skill* describes a primitive reasoning capability, without decomposing the problem to be solved into subproblems. The sort Skill (Figure 4.10) is defined as a subsort of the sort Capability. A skill does not need to introduce any new feature beyond the properties defined by the sort Capability. Therefore, a skill has pragmatics and ontologies inherited from the sort Knowledge-Component; and inputs, outputs, competence, knowledge-roles and assumptions inherited from the sort Capability.

Skill **is-a** *Capability*
pragmatics → *Pragmatics*
ontologies → set-of *Ontology*
inputs → set-of *Signature*
outputs → set-of *Signature*
competence → *Competence*
knowledge-roles → set-of *Signature*
assumptions → set-of *Assumptions*
communication → *Communication*

Figure 4.10: The Skill sort

Task-decomposers

A *task-decomposer* is a capability that decomposes a problem in subproblems. A task-decomposer capability describes how a task is decomposed into a number of subtasks which combined competence satisfies the postconditions of the capability, whenever the preconditions of the capability hold.

The sort Task-Decomposer (Figure 4.11) is defined as a subsort of the sort Capability. Like a capability, a task-decomposer is described with inputs, outputs, knowledge-roles, competence, assumptions and communication, but in addition a task-decomposer provides the *subtasks* in which it decomposes a task, and the

operational description of the problem-solving process, representing the control and data flow among subtasks.

- The *subtasks* feature specifies a collection of tasks the combined competence of which can achieve the competence defined by the task-decomposer. The subtasks feature is defined as a set of elements of the sort Task.
- The *operational description* feature specifies the reasoning steps applied by a capability in order to achieve its competence. In the Knowledge-Modelling Ontology, the operational description is defined as an element of the sort *Operational-Description*. Like the sort *Communication*, the sort *Operational-Description* is more closer to the implementation details than the other features characterizing a task-decomposer, and consequently it will be defined at the Operational Framework, as another element to be extend the KMF in order to become an Agent Capability Description Language (section 5.4.3).

Task-Decomposer **is-a** Capability
pragmatics → *Pragmatics*
ontologies → set-of *Ontology*
inputs → set-of *Signature*
outputs → set-of *Signature*
competence → *Competence*
knowledge-roles → set-of *Signature*
assumptions → set-of *Formula*
communication → *Communication*
subtasks → set-of *Task*
operational-description → *Operational-Description*

Figure 4.11: The Task-Decomposer sort

While other Knowledge Modelling frameworks describe the reasoning process of a capability in terms of inference steps, we prefer to describe complex capabilities in terms of a decomposition of interrelated subtasks (§5.4.3), without further specifying which capability is applied to solve each subtask (this is decided during the configuration of the task-decomposer at the Knowledge Configuration process). Therefore, an operational description should be understood as a template for decomposing a complex task into subtasks, without specifying the way each subtask is solved, only the way they are combined. This way of representing a task decomposition in terms of subtasks and not in terms of capabilities maximizes reuse and allows a flexible configuration of a task by assigning the most appropriate capabilities for each specified problem. A task-decomposer is then a problem decomposition schema that can be instantiated or configured on-demand, by selecting capabilities and domain-models suitable for a specific problem. This way of decomposing a problem into subtasks is a key element

of the Knowledge Configuration process backing the on-demand configuration of Multi-Agent Systems to satisfy specified problem requirements. We do not introduce now a concrete language for specifying the operational-description, since this feature, like the communication feature, pertains to the Operational Framework.

Both the *Communication* and the *Operational-Description* sorts are excluded from the Knowledge Configuration process and the Knowledge Modelling Ontology, thus they are described at the Operational Framework (Section §5.4). The idea is to keep the Knowledge Configuration process independent of the operational-level details of agent capabilities.

While a task decomposer corresponds to a *white-box* model of a capability, a skill is described using a *black-box* model. This consideration implies a weak notion of the difference between task-decomposers and skills according to the two ways of modelling a process: white-box vs black-box, i.e. choosing a skill to represent a capability does not necessarily mean that a very simple capability is being represented, but it can obey to some interest in hiding the details of a complex reasoning process. The purpose of introducing this consideration is to support a more flexible approach to the operational framework. For example, an agent may prefer to hide the details of a capability in order to keep it under local control. Therefore, autonomous agents may decide to declare a capability as a task-decomposer or as a skill according to its own preferences or to some conditions.

Domain-models

A *domain-model* (DM) specifies the concepts, relations and properties characterizing the knowledge of some application domain.

The sort Domain-Model (Figure 4.12) is defined as a subsort of the sort Knowledge-Component, and as such a domain-model has a name, a textual description, pragmatics and ontologies. Moreover, the sort *domain-model* introduces some new features, namely *knowledge-roles*, *properties* and *meta-knowledge*.

- The *knowledge-roles* define the concepts provided by the domain-model ontologies to describe domain knowledge that can be used by a capability. The knowledge-roles are specified as a set of elements of the sort Signature-element, to be refined by the sort Signature-element in the Object Language.

- The *properties* feature is used to represent properties verified by the knowledge characterized by the domain-model.

- The *meta-knowledge* feature represents properties of the knowledge that are assumed to be true though they cannot be verified.

Domain-models are used to explicitly describe the knowledge required by a capability to solve a problem. Therefore, the concepts defined by a domain-

model ontology should be understood by a capability in order to use the knowledge characterized by that domain-model, or there must exist a mapping between the concepts defined in the capability ontology and equivalent concepts in the domain-model ontology. Moreover, in addition to fill in the knowledge-roles of a capability with domain-models, the assumptions of a capability must be satisfied by the domain-models of choice. These assumptions can be verified by either the domain-model's properties or the meta-knowledge.

Domain-Model **is-a** *Knowledge-Component*
pragmatics → *Pragmatics*
ontologies → set-of *Ontology*
knowledge-roles → set-of *Signature-element*
properties → set-of *Formula*
meta-knowledge → set-of *Formula*

Figure 4.12: The Domain-Model sort

Ontologies

Concerning ontologies, we agree with [Guarino, 1997b] about the potential role of explicit ontologies to support reuse. Although a definition of what ontologies are is still a debated issue, it is a topic of active research in the AI community, and has been declared as a key issue in maximizing reuse [Fensel, 1997a, Fensel and Benjamins, 1998b]. From that view, the main goal of an ontology is to make knowledge explicit and sharable. Below follows a concrete definition that expresses worth the role played by ontologies in our framework.

> Ontologies are shared agreements about shared conceptualizations. Shared conceptualizations include conceptual frameworks for modelling domain knowledge; content-specific protocols for communicating among interoperating agents; and agreements about the representation of particular domain theories. In the knowledge sharing context, ontologies are specified in the form of definitions of representational vocabulary [Guarino, 1997b].

In ORCAS, ontologies are used to explicitly declare the concepts used to specify the features characterizing a component. In particular, any concept should be specified as a refinement (a subsort) of two sorts provided by the Object Language: *Signature-element* and *Formula*. We adopt the usual approach to represent ontologies as hierarchies of concepts and relations. Specifically, we use the *Feature Terms* [Armengol and Plaza, 1997, Arcos, 1997] formalism to describe the sorts defined by an ontology, as described in §4.3.

In Feature Terms, concepts are organized as a hierarchy of *sorts*, and both descriptions and individuals as represented as collections of functional relations.

Any ontology used to specify components in ORCAS provides two basic sorts from which all the other sorts are descendant: the sort *Signature-element* and the sort *Formula*.

The sort Ontology is defined as a subsort of the sort Knowledge-Component (Figure 4.13), and as such it inherits the pragmatics feature, and the ontologies feature. But the information characterizing an ontology is mainly provided by a hierarchy of sorts below two main sorts: *Signature-element* and *Formula*. Therefore, an ontology includes a collection of sorts defined as subsorts of *Signature-element*, and a collection of sorts defined as subsorts of *Formula*. Furthermore, an element (an instance) of the sort *Ontology* can import other ontologies through the "ontologies" feature, inherited from the sort Knowledge-Component, and specified as a collection of elements of sort Ontology.

Ontology **is-a** *Knowledge-Component*
pragmatics → *Pragmatics*
ontologies → set-of *Ontology*
signature → set-of *Signature-element*
formulae → set-of *Formula*

Figure 4.13: The Ontology sort

Figure 4.14 shows the sorts defined by the ontology used to describe the tasks and capabilities in the WIM application (Chapter 7). The sort *Signature-element* is used to describe the inputs and outputs of a task or capability, and the knowledge-roles of a capability or a domain-model. The sort *Formula* is used to specify the preconditions and postconditions of a task or capability, the assumptions of a capability, and the properties and meta-knowledge of a domain-model.

The explicit, declarative and shared nature of component's ontologies make them appropriate to annotate components with semantic information; thus enabling to compare component specifications on a *semantic matching* basis [Guarino, 1997a, Paolucci et al., 2002]. Semantic matching has been defined as an operator that takes two graph-like structures (e.g., database schemas or ontologies) and produces a mapping between elements of the two graphs that correspond semantically to each other [Giunchiglia and Shvaiko, 2003].

4.2.2 Matching relations

This section describes the different relationships that may be established between components in the Abstract Architecture and the role these relations play in the Knowledge Configuration process.

The ORCAS KMF describes a system as a collection of tasks, capabilities and domain-models. Nevertheless, an enumeration of classes of components is not enough to describe a system, some structuring principle is needed. In

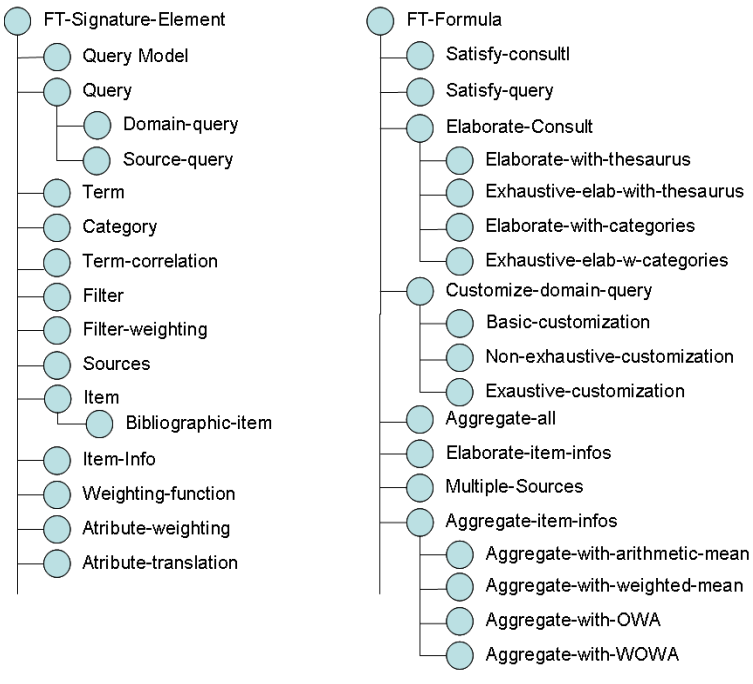


Figure 4.14: Hierarchy of sorts in the ISA-Ontology (from the WIM application, Chapter 7)

ORCAS this structure derives from the functional relations that can be established among components in the Abstract Architecture. Specifically, the ORCAS KMF includes two types of binary relations between components, namely *Task-Capability matching* and *Capability-Domain matching*.

- A *Task-capability matching* relation is defined between a task and a capability. Intuitively, a task-capability matching denotes a *suitability* relation: a task-capability relation is verified (is evaluated as true) when the capability is suitable for the task. In other words, a task "matches" a capability if the capability is able to solve the type of problems defined by the task. This relation compares the inputs, outputs and competence of a task against the homonym features of a capability to determine wether the application of the capability is able to achieve the postconditions of the task, whenever the preconditions of the task hold.
- A *Capability-domain matching* relation is defined between a capability and a collection of domain models characterizing the application domain. Since a capability may include many knowledge-roles, then a domain-model would be required to fill in each knowledge-role. Intuitively, a capability-domain matching denotes a relation of *satisfiability*: a capability "matches" a set of domain-models when the knowledge characterized by those domain-models satisfies the knowledge requirements (the *assumptions*) of the capability for each knowledge-role. This relation is defined in terms of knowledge-roles and capability assumptions that are satisfied by the properties and meta-knowledge of the domain-models.

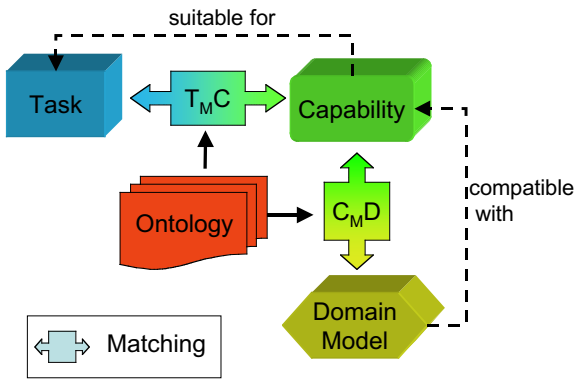


Figure 4.15: Matching relations in the Abstract Architecture

Figure 4.15 shows the components in the Abstract Architecture and the matching relations than can be established among components.

Understanding the matching relations is important because the operations to be performed during the Knowledge Configuration process are based on these

relations, and also because these relations are necessary to drive the analysis and the knowledge acquisition phases of software development.

The second point will be realized progressively through the rest of the chapter, but the main idea is using the matching relations to constrain the selection of components during the Knowledge Configuration process.

A matching relation (also referred as a "match") is determined by comparing two specifications, and has the goal of determining whether two software components are related in some way, e.g. two software components "match" if they are substitutable or if one component can be adapted to fit the requirements of another one.

The definition of a matching relation between components is built upon the definition of a more basic relation between component features. Since component features are specified in terms of the Object Language, matching will be defined upon a relation between elements (signature-elements and formulae) of the Object Language. Hence, in order to maximize the reuse of the Abstract Architecture over different Object Languages, we introduce two levels in the definition of a matching relation: the *abstract-level matching* and the *object-level matching*.

- The *abstract-level matching* is situated at the level of the Abstract Architecture. Matching relations at this level are based on an *abstract relation* between component features. Therefore, any system using the ORCAS Abstract Architecture can use the matching relations at the abstract level.

- The *object-level matching* is concerned with the Object Language. Matching relations at this level are defined as a refinement of the matching relations at the abstract level. This refinement is achieved by replacing the abstract relation by an *object relation* that is defined between elements (signature-elements and formulae) in the Object Language.

Before providing specific definitions of the ORCAS matching relations we will give some basic definitions concerning matching. Our approach to component matching is based on a combination of *signature matching* [Zaremski and Wing, 1995] and *specification matching* [Zaremski and Wing, 1997], that we prefer to call *competence matching*. Signature matching relations compare the interface of two components in terms of the types of information (and knowledge) they use (inputs and knowledge-roles) and produce (output). Competence matching relations compare the features characterizing the competence (preconditions and postconditions) of two components to determine if two components are substitutable, or if a component satisfies the requirements of another (e.g. a capability that is suitable for a task must satisfy all postconditions of that task).

In general, a matching relation is defined between two component specifications as follows:

$$Match(S, S') = match_{sig}(S, S) \wedge \ match_{comp}(Q, S)$$

where S, S' are the specification of two components, $match_{sig}$ is a *signature matching* and $match_{comp}$ is a *competence matching*.

The particular definition of *signature matching* and *competence matching* will be different for the different types of matching relation (Task-Capability matching or Capability-Domain matching).

Task-capability matching

A *Task-capability match* is defined between a task T and a capability C to determine wether C is suitable for the T (i.e. C can be applied to solve the type of problems characterized by T).

We define a *Task-capability match* as the conjunction of a *Generalized Type Match* over the input signature specification, and a *Specialized Type Match* over the output signature specification [Zaremski and Wing, 1995], and a *Plug-in Match* [Zaremski and Wing, 1997] over the competence specification.

Definition 4.1 ***(Task-capability match)***

$$match(T, C) = match_{gen}(T_{in}, C_{in}) \wedge match_{spec}(T_{out}, C_{out}) \wedge match_{plugin}(T, C)$$

where T is a task C and is a capability, $match_{gen}$ is a *Generalized Signature Match*, $match_{spec}$ is a *Specialized Signature Match*, and $match_{plugin}$ is a *Plug-in match* defined over the competence specification (preconditions and postconditions).

The *Generalized Signature Match* over the input signatures means that the capability has an input signature C_{in} equal or more general than task input signature T_{in}.

$$match_{gen}(T_{in}, C_{in}) = \ (T_{in} \geq C_{in})$$

Inversely, the *Specialized Signature Match* over the output signatures means that the capability output is of the same type or of a more specific type than the task output.

$$match_{spec}(T_{out}, C_{out}) = \ (T_{out} \leq C_{out})$$

This combination of generalized and specialized matchings has the following justification:

- On the one hand, a capability with a more general input than a task implies the capability can extract from the task input all the information it requires. However, if we select a capability with an input more specific than a task (with more information), then the capability cannot obtain all the information it uses as input, and this fact could result on a bad capability operation.
- On the other hand, a capability with an output signature more specific means that it is able to provide all the information characterizing the output signature of a task, which is not true if the output of the capability is more general (with less information) than the output of the task.

Moreover, the *Plug-in Match* ($match_{plugin}$) [Zaremski and Wing, 1997] requires a capability C to have equal or weaker preconditions than a task T and equal or stronger postconditions than T:

$$match_{plugin}(T, C) = \ (T_{pre} \Rightarrow C_{pre}) \wedge (C_{post} \Rightarrow T_{post})$$

The reason to use that kind of matching is the following: we want to use capabilities that are suitable for (able to solve) a task, thus we want that whenever the preconditions specified by the task hold, the application of the capability guaranteed that the postconditions of the task will hold after its application.

The demonstration of the former property from the definition of the plug-in match is as follows: T_{pre} entails that C_{pre} holds, due to the first conjunct of the Plug-in Match, and C guarantees that $C_{pre} \Rightarrow C_{post}$; consequently, it's assured that C_{post} will hold after executing C, which entails also T_{post}, due to the second conjunct ($C_{post} \Rightarrow T_{post}$).

These definitions will be refined later in terms of object-level matching relations on signatures and on formulae (§4.3.3).

Capability-domain matching

Matching of domain-models and capabilities is slightly different, since a domain-model does not include a competence specification, neither an input nor an output. However, a capability may introduce more than one knowledge-role, consequently many domain-models would be required to match a single capability.

If a capability introduces only one knowledge-role, then we can say that a capability matches a domain model when the domain-model provides the kind of knowledge characterized by that knowledge-role, and satisfies the assumptions established by the capability for that knowledge-role. Moreover, if a capability specifies more than one knowledge-role, then for each knowledge-role there should exist one domain-model matching the specification of the capability .

Let's define the set of knowledge-roles of a capability as $C_{kr} = \{C^i_{kr} : i = 1 \ldots n\}$, and let's represent the set of assumptions of a capability over a particular knowledge-role as C^i_{asm}. A matching relation between a knowledge-role of a capability ($C^i_{kr} \in C_{kr}$) and one domain-model (M) is called a *partial capability-domain match*. A partial capability-domain match is defined such that there is at least one knowledge-roles in the domain-model (M_{kr}) that is equivalent or more specific than the knowledge-role of the capability (C^i_{kr}), and the assumptions of the capability for that knowledge-role C^i_{asm} are satisfied by the union of the properties and meta-knowledge of the domain-model ($M_{prop} \cup M_{mk}$), namely

$$Match_{partial}(C, M, C^i_{kr}) = \ match_{esp}(C^i_{kr}, M_{kr}) \wedge (M_{prop} \cup M_{mk} \Rightarrow C^i_{asm})$$

where C^i_{kr} is a knowledge-role of a capability and M is a domain-model; $match_{spec}$ is a *Specialized Signature Match*; M_{prop} and M_{mk} are the *properties* and *meta-knowledge* of M, and C^i_{asm} are the assumptions of capability C for the knowledge-role C^i_{kr}.

In this definition, a partial *capability-domain match* is expressed as a combination of a *Specialized Type Match* between the signature specification of a capability knowledge-role and the specification of the knowledge-roles of a domain-model, and a special kind of *Plug-in Match* defined between the specification of the assumptions of a capability for a single knowledge-role, and the properties and meta-knowledge of the domain-model.

The *Specialized Type Match* between a knowledge-role and a domain-model is defined as follows:

$$match_{esp}(C^i_{kr}, M_{kr}) = (C^i_{kr} \leq M_{kr})$$

The reason to use a *Specialized Type Match* here is that we must ensure the knowledge-roles characterized by a domain-model M can provide at least all the information required by a knowledge-role of a capability C^i_{kr}. This condition is guaranteed when the signature specification of the knowledge-roles of the domain-model (M_{kr}) is equal to or specializes the signature specification of the capability knowledge-role (C^i_{kr}). If M_{kr} was more general than C^i_{kr}, then it may occur than some of the information required by C^i_{kr} cannot be provided by the knowledge characterized by M_{kr}, and thus the capability cannot use that knowledge appropriately.

Moreover, in order for a capability to use the information characterized by a knowledge-role (C^i_{kr}), the domain-model providing that knowledge-role should guarantee that the assumptions of the capability over that role (C^i_{asm}) are satisfied by M. The specification of a domain-model is divided in two parts called *properties* (M_{prop}) and *meta-knowledge* (M_{mk}), consequently we define that the assumptions of a capability for a knowledge-role C^i_{asm} are satisfied by a domain-model when these assumptions are implied by the union of both the properties and meta-knowledge of the domain-model ($M_{prop} \cup M_{mk}$), as follows:

$$(M_{prop} \cup M_{mk} \Rightarrow C^i_{asm})$$

Now we can define a matching relation between a capability and a collection of domain models $\mathcal{M}$ that satisfy C as a conjunction of matching relations between pairs consisting of a knowledge-role (a signature element) and a domain model that matches it, such that there is a domain-model matching every knowledge-role specified by the capability.

Definition 4.2 ***(Capability-domain match)***

$match(C, \mathcal{M}) = \forall C^i_{kr} \in C_{kr} : \exists M \in \mathcal{M} | Match_{partial}(C, M, C^i_{kr})$
$= \forall C^i_{kr} \in C_{kr} : \exists M \in \mathcal{M} |\ (C^i_{kr} \leq M_{kr}) \wedge (M_{prop} \cup M_{mk} \Rightarrow C^i_{asm})$

where C is a capability, $\mathcal{M}$ is a set of domain-models; $match_{spec}$ is a *Specialized Signature Match*; M_{prop} and M_{mk} are the *properties* and *meta-knowledge* of M, and C_{asm} are the assumptions of capability C.

Ontology mappings

Since ORCAS components declare its conceptualizations (the "universe of discourse") as ontologies, and two components may be declared using different ontologies, then it can be necessary to establish a mapping between the concepts of the two ontologies, what is called a *ontology mapping*.

An *ontology mapping* is a declarative specification of the transformations required to match elements of one ontology to elements of another ontology. An example of a mapping is a *renaming*, but a mapping can include any kind of syntactic or semantic transformation: *numerical mapping*, *lexical mapping*, *regular expression mapping* and others [Park et al., 1998].

Nevertheless, in this thesis we focus on the matchmaking process (the process of verifying if a matching relation holds) assuming that all the components share the same ontologies or the required ontology mappings are already built.

4.3 The Object Language

This section presents a specific Object Language to be used within the ORCAS KMF to specify component features, and a particular relation (subsumption) to compare the specification of components in order to verify whether a matching relation holds.

We need an Object Language $\mathbb{O}$ in which to write the specification of capabilities, tasks and domain-models. Moreover, the Object Language has to be used also to represent other objects required by the Knowledge Configuration process, including the specification of problem requirements, task-configurations (the result of the Knowledge-Configuration process), and search states (Knowledge Configuration is approached as a search process over the space of possible configurations).

The Object Language provides a formalism for defining the terminologies used to specify component features. As we have introduced when describing the Abstract Architecture (§4.2.1), most of the features used to describe a component are specified by either signature elements or formulae in the Object Language: The sort SIGNATURE-ELEMENT is used to describe the input and output of tasks and capabilities, and the knowledge-roles of capabilities and domain-models as well. The sort FORMULA is used to specify the preconditions and postconditions of tasks and capabilities, the assumptions of a capability, and the properties and meta-knowledge of a domain-model[1]. In ORCAS the terminologies used to specify component features are represented by ontologies, thus the elements of the Object Language are specified as sorts of some ontology.

Figure 4.14 shows an example of an Object Language onotology used to specify tasks and capabilities in the WIM application (Chapter 7). Notice that

[1] Usually, knowledge-modelling languages describe these kind of specifications with a formal language, some examples are LOOM [Gaspari et al., 1998], OCML [Motta, 1999] and KARL [Studer et al., 1996]

every sort is a specialization (a subtype) of either the sort Signature-element or the sort Formula.

Many languages could be used as the Object Language while keeping the ORCAS Abstract Architecture, thanks to a clear separation of two levels in the definition of a matching relation: the *abstract level* matching and the *object level* matching.

We have already defined the matching relations at the *abstract level*, now we are going to describe the *Feature Terms* formalism to be used as the Object Language. Following, the ORCAS matching relations will be rewrited using subsumption between feature terms as the object-level matching relation..

4.3.1 The Language of Feature Terms

The ORCAS Object Language is based on the Feature Terms formalism to represent ontologies and component features. *Feature Terms* (also called feature structures or ψ-terms) are a generalization of first order terms. The difference between feature terms and first order terms is the following: a first order term, e.g. $f(x, y, g(x, y))$ can be formally described as a tree and a fixed tree-traversal order. In other words, parameters are identified by position. The intuition behind a feature term is that it can be described as a labelled graph i.e. parameters are identified by name.

Specifically, we use a concrete implementation of feature terms as embodied in the NOOS representation language [Arcos, 1997], in which several Case Based Reasoning systems have been described and implemented [Arcos et al., 1998, Arcos, 2001]. This formalism organizes concepts into a hierarchy of *sorts*, and represents descriptions and individuals as collections of features (functional relations) called *feature terms*. The attributes used to describe a component (signature-elements and formulae) will be specified as feature terms in the Feature Terms formalism.

Before to define Feature Terms formally we need to introduce the following elements:

1. a signature $\Sigma = \langle \mathcal{S}, \mathcal{F}, \leq \rangle$ (where $\mathcal{S}$ is a set of sort symbols that includes $\perp$; $\mathcal{F}$ is a set of feature symbols; and $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\perp$ is the least element),

2. and a set ϑ of variables;

Succinctly, we can now define a feature term ψ as an expression of the form:

$$\psi ::= X : s[f_1 \doteq \Psi_1 \ldots f_n \doteq \Psi_n] \tag{4.1}$$

where X is a variable in ϑ that is called the *root* of the feature term, s is a sort in $\mathcal{S}$ (the root sort), $f_1 \ldots f_n$ are features in $\mathcal{F}$, $n \leq 0$, and each Ψ_i is a set of Feature Terms and variables. When $n = 0$ we are defining a term without features. The set of variables occurring in ψ is noted as ϑ_ψ.

Sorts have an informational order relation ($\leq$) among them, where $\psi \leq \psi'$ means that ψ has less information than ψ' (or equivalently that ψ is more general than ψ'). The minimal element ($\bot$) is called *any* and represents the minimum information. A feature with an unknown value is represented as having the value *any*. All other sorts are more specific than *any*.

4.3.2 Subsumption

A basic notion of the Feature Terms formalism is that of *subsumption*, which we use as the inference mechanism. Subsumption is an order relation among terms built on the top of the $\leq$ relation among sorts.

Intuitively, we say of two Feature Terms $\psi, \psi' \in \Phi$ that ψ subsumes ψ' ($\psi \sqsubseteq \psi'$) when all that is true for ψ is also true for ψ'. A more formal definition of subsumption is introduced below, but first we need to introduce some notation and basic definitions:

- $Root(\psi)$ is defined as a function that returns the root of a term (a variable X).
- $Sort(X)$ is defined as a function that returns the sort of the variable X.
- A *path* $\rho(X, f_i)$ is defined as a sequence of features going from the variable X to the feature f_i. There is a *path equality* when two paths point to the same value. Path equality is equivalent to variable equality in first order terms.
- We use a "dot notation" to reference a particular feature within a term, thus $\psi.f$ refers to the feature f of the term ψ.

Now we can define subsumption as follows:

Definition 4.3 *(Subsumption) Feature term ψ subsumes ψ' ($\psi \sqsubseteq \psi'$) if:*

1. $Sort(Root(\psi)) \leq Sort(Root(\psi'))$, *i.e. the root sort of ψ' is the same sort or a subsort of the root sort of ψ.*
2. $\forall f \in \mathcal{F} : \psi.f \neq \bot \Rightarrow \psi'.f \neq \bot$, *i.e. every defined feature in ψ is also defined (has a value different from $\bot$) in ψ'.*
3. $\forall f \in \mathcal{F} : \psi.f = v \neq \bot \Rightarrow v \sqsubseteq v' = \psi'.f$ *if v is a singleton,*
4. *otherwise if $\psi.f$ is a set $\psi.f = V = \{v_1 \ldots v_m\}$ then $\forall v_i \in V : \exists v'_j \in V' = \psi'.f : v_i \sqsubseteq v'_j)$, i.e. there is a subsumption mapping between the sets.*
5. $(\rho(Root(\psi), f_1) = \rho((Root(\psi), f_2) = v) \Rightarrow (\rho(Root(\psi'), f_1) = \rho((Root(\psi'), f_2) = v' \wedge v \sqsubseteq v')$*; i.e., path equality is satisfied downwards.*

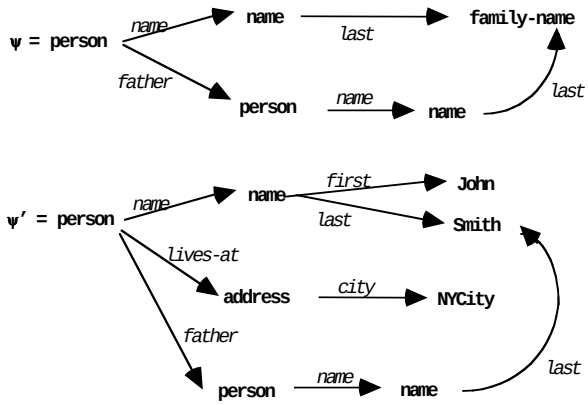


Figure 4.16: Representation of feature terms as labelled graphs

Notice that step 4 of Definition 4.3 provides a concrete interpretation of the subsumption between two sets: thus given two sets of feature terms Ψ, Ψ' we say that $\Psi \sqsubseteq \Psi'$ if step 4 of Definition 4.3 is verified.

Figure 4.16 shows an example of two features that verify a subsumption relation. This picture shows feature terms as labelled directed graphs: for each variable $X : s$ there is a node q labelled with sort s, while an arc from q to another node q' is labelled by f, for each feature f defined in q with feature value q'.

In this example there are two feature terms ψ, ψ' with the same root sort, person. Notice that ψ subsumes ψ', $\psi \sqsubseteq \psi'$, since ψ' contains more information than ψ (ψ' is a specialization of ψ, which entails that all that is true for ψ' is also true for ψ, $\psi \Rightarrow \psi$): Both ψ and ψ' are of sort person. ψ has two features, *name* and *father*, specified as elements of sort name and person respectively. The feature *name* is specified as terms of sort name, which has a feature called *last* representing the father's name. For both terms the father's name has a single feature called *name* which is the same (there is path equality) than the *last* name of the person. However, ψ' specializes ψ in the following aspects: the *last* name of ψ is specified as a term of sort family-name, and the last name of ψ' is John, which identifies a term of sort family name, and thus it is just a specialization of the last name of ψ'. Finally, ψ' has another feature not existing in ψ (*lives-at*) that contains a partial description of their home address. ψ' is a specialization of *psi*, since ψ' contains more information than ψ, and thus we conclude than ψ' is subsumed by ψ.

4.3.3 Matching by subsumption

Subsumption can be now used to define the matching relations at the object-level. Both signatures and formulae in our Object Language are represented using *terms* in the Feature Terms formalism. Moreover, each term ψ has an associated root sort ($Sort(\psi)$), and these sorts are organized into a hierarchy of sorts in the component's ontology. As a consequence, terms without any feature can be directly evaluated upon the informational partial order relation ($\leq$) over their root sorts.

In general a signature or a formula can be expressed as a full fledged Feature Terms structure, and therefore the relations ($\leq$ and $\Rightarrow$) used to define the matching relations should be reformulated using Feature Terms concepts, namely the informational partial order relation $\leq$ and the implication $\Rightarrow$ relation. These relations will be specialized by the subsumption relation among Feature Terms (Definition 4.3) as follows.

- On the one hand, if a term ψ subsumes another term ψ' ($\psi \sqsubseteq \psi'$), then the root sorts for these terms verify a partial order relation (Step 1 of Definition 4.3): $\psi \sqsubseteq \psi' \Rightarrow Sort(\psi) \leq Sort(\psi')$. Intuitively, the subsumption relation can be regarded as a generalization of the informational order relation ($\leq$) to compare terms having no empty features (features with a value different of $\perp$) with structures, in place of single sorts. Therefore, the partial order relation ($\leq$) introduced at the abstract-level matching is replaced by the subsumption relation at the object-level matching.

- On the other hand, the implication relation ($\Rightarrow$) introduced at the abstract-level matching boils down to subsumption over Feature Terms at the object-level matching. Intuitively, a term ψ subsuming another term ψ' ($\psi \sqsubseteq \psi'$) means that ψ' is more specific or has more information than ψ, and this implies that all that is true for ψ is also true for ψ': $\psi \Rightarrow \psi'$.

Now we can express signature matching over inputs and outputs using *subsumption* over Feature Terms, as follows:

$$match_{sig}(T, C) = T_{in} \sqsubseteq C_{in} \wedge T_{out} \sqsubseteq C_{out}$$

where subsumption ($\sqsubseteq$) is among sets (step 4 of Definition 4.3).

Similarly, we can reformulate the definition of a *Plug-in Match* between a task and a capability as follows:

$$match_{plugin}(T, C) = T_{pre} \sqsubseteq C_{pre} \wedge C_{post} \sqsubseteq T_{post}$$

Now that we have defined the basic matching relations between signature-elements and formulae expressed at the object-level, we are ready to reformulate the whole definition of a Task-capability match (Definition 4.1) to the language of Feature Terms using subsumption:

Definition 4.4 ***(Task-capability match by subsumption)***
$match(T, C) = T_{in} \sqsubseteq C_{in} \wedge C_{out} \sqsubseteq T_{out} \ \wedge C_{pre} \sqsubseteq T_{pre} \wedge T_{post} \sqsubseteq C_{post}$

Intuitively, this definition is justified as follows: If the input of the capability subsumes the input of the task ($C_{in} \sqsubseteq T_{in}$) then the input for the task will provide at least all the information required by the capability, since the input of the task is more specific or has more information that the input of the task. Complementarily, the output of the capability subsumes the output of task ($C_{out} \sqsubseteq T_{out}$) implies that the output of the capability is more specific than the output of the task and, consequently, the application of the capability can provide at least all the information required by the task output.

Concerning the competence, the fact that the preconditions of the capability subsume the preconditions of the task ($C_{pre} \sqsubseteq T_{pre}$) means that all the preconditions of the capability are guaranteed by the equal or more specific preconditions of the task, and viceversa: the fact that the postconditions of the task subsume the capability postconditions ($T_{post} \sqsubseteq C_{post}$) means that all task postconditions are guaranteed by the capability postconditions.

Similarly, we can specialize the definition of a Capability-domain match (Definition 4.1) to the language of Feature Terms by using subsumption as the basic matching:

Definition 4.5 ***(Capability-domain match)***

$$match(C, \mathcal{M}) = \forall C^i_{kr} \in C_{kr} : \exists M \in \mathcal{M} | Match_{partial}(C, M, C^i_{kr})$$
$$= \forall C^i_{kr} \in C_{kr} : \exists M \in \mathcal{M} |\ (C^i_{kr} \sqsubseteq M_{kr}) \wedge (C^i_{asm} \sqsubseteq M_{prop} \cup M_{mk})$$

where C is a capability, $\mathcal{M}$ is a set of domain-models; C_{kr} are the knowledge-roles, and C^i_{asm} are the assumptions of the capability for the ^ith knowledge-role, specified as a set of signature-elements and formulae respectively; and M_{prop}, M_{mk} are the properties and meta-knowledge of a domain-model, specified as formulae. Both signature-elements and formulae are specified by feature terms.

In this definition, two conditions are imposed for establishing a match between a capability and a collection of domain-model:

1. the specification of knowledge-role signatures by the domain-models must provide at least as much information as required by the signature specification of the knowledge-role of the capability. This condition is ensured if for each signature specifying a knowledge-role of the capability, there is a specification of domain-model signatures that refines or is more specific than it, which in feature terms is expressed by the idea of being subsumed ($C^i_{kr} \sqsubseteq M_{kr}$);

2. the assumptions of the capability for each knowledge-role C^i_{kr} must be satisfied by the properties and meta-knowledge of a domain-model that in addition verifies the matching between the signature specification: this is assured if all that is true for the domain-model is also true for the assumptions of the capability ($(M_{prop} \cup M_{mk}) \Rightarrow C^i_{asm}$). Using feature terms this condition can be expressed using subsumption as $C_{asm} \sqsubseteq (M_{prop} \cup M_{mk})$.

4.4 Knowledge Configuration

We have defined the MAS configuration process as being performed at two layers: the Knowledge Configuration process, that is situated at the knowledge-layer (this chapter), and the Team Formation process, that is carried on at the operational-layer (Chapter 5).

During the previous sections of the chapter the Abstract Architecture and a concrete Object Language for the ORCAS KMF have been described. In addition, we have formally defined the matching relations to be used during the Knowledge Configuration process in order to verify whether a capability is suitable for a task, and whether a domain-model satisfies a capability. Hence, we have described the basic concepts required to explain the Knowledge Configuration process itself, which is the aim of this section.

The *Knowledge Configuration* process has the goal of finding a configuration in terms of a composition of application tasks, agent capabilities and domain-models, in such a way that the requirements of the problem at hand are satisfied. This process actually follows a *Problem Specification* process, the main purpose of which is the characterization of the problem at hand in order to later select the most appropriate components during the Knowledge Configuration process. This characterization of a problem is formalized by a specification of *problem requirements* to be satisfied by a Task-Configuration.

After introducing some basic definitions and notation (§4.4.1) we will describe the Problem Specification process (§4.4.2) and the Knowledge Configuration process (§4.4.3). Next, three strategies for the Knowledge Configuration process (§4.4.4) are presented.

4.4.1 Notation and basic definitions

We have already seen the components defined in the Abstract Architecture and the different relations constraining the way in which components can be connected, which have been defined as matching relations. This section summarizes the specification of components in the Abstract Architecture and introduces some basic definitions.

We deal with three main types of knowledge-level components, namely: *task*, *domain-model* and *capability*, which have two further subtypes: *skill* and *task-decomposer*.

Knowl.-Component	X	
Task	$T < X$	$T = \langle in, out, pre, post \rangle$
Capability	$C < X$	$C = \langle in, out, pre, post, com, asm, kr \rangle$
Task-Decomposer	$D < C < X$	$D = \langle in, out, pre, post, com, asm, kr, st \rangle$
Skill	$S < C < X$	$S = \langle in, out, pre, post, com, asm, kr \rangle$
Domain Model	$M < X$	$M = \langle kr, prop, mk \rangle$

Table 4.1: Types of knowledge components an their main features

Table 4.1 sums up the hierarchy of sorts used to describe the components in the Abstract Architecture, and the features used to specify each component, where $A < B$ means that A is a subsort (subtype) of B, st are subtasks ($st \subset \mathcal{T}$), in, out, kr are inputs, outputs and knowledge-roles, specified as signature-elements in the Object Language $\mathbb{O}$, $pre, post, asm$ are preconditions, postconditions and assumptions, specified as formulae in the same language $\mathbb{O}$, com, od are the communication aspects and the operational description of a capability respectively, and pro, mk are the properties and the metaknowledge of a domain model, specified by formulae in $\mathbb{O}$.

We will note an element of a tuple specification as subscript, e.g. T_{in} is the input signature of task T and T_{post} are the postconditions of T.

However, since the Knowledge Configuration process requires a repository of components as an input to choose from the components of a task-configuration, we will introduce the idea of a repository of components or a *library*.

A *Library* is a collection of tasks and capabilities specified using some Object Language. A Library is independent of the domain because both tasks and capabilities are described in terms of their own ontologies, and not in terms of the domain ontology.

Definition 4.6 ***(Library)***
$\mathbb{L} = \langle \mathcal{T}, \mathcal{C}, \mathbb{O} \rangle$,
where

- $\mathcal{T}$ *is a set of tasks,*
- $\mathcal{C}$ *is a set of capabilities,*
- *and* $\mathbb{O}$ *is the Object Language.*

In the following definitions we will note $\mathcal{T}$ and $\mathcal{C}$ as the set of tasks and capabilities in the library used by the Knowledge Configuration process.

The Knowledge Configuration process takes a specification of stated problem requirements and a library of components as input and produces a task-configuration as output. Since a task-configuration is a complex structure we need first to define its constituent elements, called configuration schemas. We will note $\kappa \in \mathbb{k}$ as a configuration schema and the set of all configuration schemas; moreover, we will note $(T \doteq U) \in \mathcal{B}$ as a *binding* and the set of all bindings, where a binding is a link between a task and a capability or a configuration schema that is selected to solve that task. More formally:

Definition 4.7 ***(Binding)*** *A binding $(T \doteq U)$ is a pair with a task $T \in \mathcal{T}$ in the* head *and a capability $C \in \mathcal{C}$ or a configuration schema $k \in \mathbb{k}$ in the* tail*: $U \in \mathcal{C} \cup \mathbb{k}$*

Definition 4.8 ***(Configuration schema)*** *A configuration schema $\kappa \in \mathbb{k}$ is a pair $\langle (T \doteq C), \{(T_i \doteq \kappa_{j_i})\}_{i=1\ldots n} \rangle$ where $T, T_1, \ldots, T_n \in \mathcal{T}$, $C \in \mathcal{C}$, and $\kappa_{j_1}, \ldots, \kappa_{j_n} \in \mathbb{k}$, and $T_1, \ldots, T_n \in C_{st}$.*

A configuration schema specifies in the *head* of the pair a binding between a task T and a capability C ($T \doteq C$). The *tail* of the configuration schema is a set of bindings from C_{st} (the subtasks of C) (which will be empty if C is a skill, since skills have no subtasks) to other configuration schemas. A configuration schema can be complete or partial, defined as follows:

$$Complete(\kappa) \Leftrightarrow \forall T_i \in C_{st}\ \exists \kappa_{j_i} : (T_i \doteq \kappa_{j_i}) \in tail(\kappa)$$

i.e. if all subtasks of C are bound to another schema in the tail; otherwise κ is *partial*.

We define a *configuration relation* $\mathbb{R}$ among schemas as follows:

$$\mathbb{R}(\kappa, \kappa') \Leftrightarrow \exists (T_i \doteq \kappa') \in tail(\kappa)$$

i.e. two schemas are related if one of them is bound to a subtask in the tail of the other.

Noting $\mathbb{R}^*$ the closure of $\mathbb{R}$ we can now define a *task-configuration* as follows:

Definition 4.9 ***(Task-configuration)*** *A task-configuration is defined in terms of configuration schemas $Conf(\kappa) = \{\kappa' \in \Bbbk | \mathbb{R}^*(\kappa, \kappa')\}$. A task-configuration $Conf(\kappa)$ can be complete or partial: $Complete(Conf(\kappa))$ iff $\forall \kappa' \in Conf(\kappa) : Complete(\kappa')$; otherwise $Conf(\kappa)$ is partial.*

Thus, a *task-configuration* is a collection of interrelated configuration schemas (starting from a root schema κ). A task-configuration is *complete* when all schemas belonging to it are complete.

We will note K a task-configuration and $\mathcal{K}$ the set of all the task-configurations ($K \in \mathcal{K}$).

4.4.2 The Problem Specification process

The Problem Specification process aims at characterizing the problem to be solved in terms of a task to be solved and a set of requirements the configured system must comply to. We are not interested on the particular strategy to carry on this process; actually we are interested in the result of the Problem Specification process, since this is an input (the other input is a library of components) for the Knowledge Configuration process (Figure 4.17). Nonetheless, we will succinctly describe a iterative approach to specify problem requirements by interacting with the user and using a matching relation between problem requirements and tasks to find out which tasks are representative of the problem at hand.

The Problem Specification process starts with the specification of some initial requirements that are used to select the *application task* T_0. The idea is that a configured system is an application that results of assembling existing components such that the resulting composition satisfies the requirements of the problem at hand. We call this class of application that is designed and assembled on-demand by reusing and existing components over a (probably new)

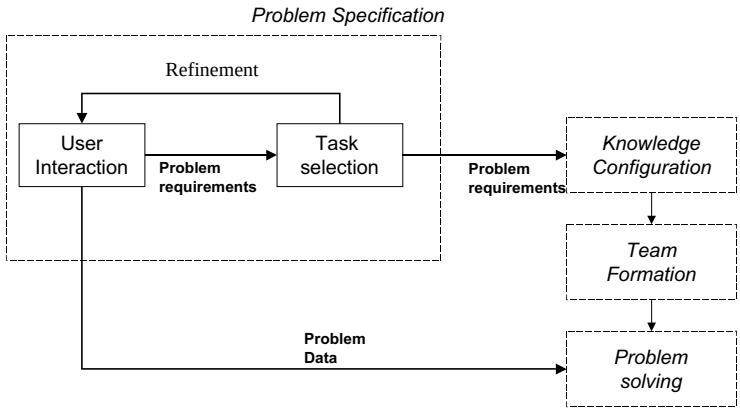


Figure 4.17: Problem Specification process

specific domain, an *on-the-fly* application. The task that is decided as the better characterization of the problem at hand is the starting point for the Knowledge Configuration process and is called the *application task*.

Since many tasks may satisfy the initial problem requirements, these requirements can be further refined and modified to better characterize the problem at hand, according to the needs and preferences of the requester (a human user or a software agent).

Problem requirements are used by two processes: Problem Specification and Knowledge Configuration. On the one hand, problems requirements are used as both input and output of the Problem Specification process; on the other hand the final output of the Problem Specification process is a collection of requirements that becomes an input for the Knowledge Configuration process.

The Problem Specification process starts with a query containing some problem requirements, and proceeds by selecting a set of tasks characterizing the problem at hand. Afterwards the user may select a task he thinks is a better characterization of his problem (the application task T_0) and may add or delete requirements from the query to be satisfied by the configuration of that task.

Definition 4.10 ***(Query)*** *A query Q is a tuple $Q = \langle in, out, pre, post, dm \rangle$*

where in, out are the query input and output signatures, which specify the type of the data provided to solve a problem, and the type of data expected as a result to that problem; pre are preconditions, which characterize the circumstances holding before starting the problem solving process, $post$ are postconditions, or the effects to bring about after solving the problem; and $dm \subset \mathcal{M}$ is a set of domain-models characterizing the application domain..

Notice that the specification of a query is like the specification of a task, but a query adds in a collection of domain-models characterizing the application domain.

In order to find out which tasks in the library satisfy the requirements stated by a query, it is necessary to establish a matching relation between a query Q and a task ($T \in \mathcal{T}$). We define a *query-task matching* relation following the idea of a *task-capability match* (Def. 4.4), but in this case the query plays the role of the task, and the task plays the role of the capability.

$$match(Q,T) = match_{gen}(Q_{in}, T_{in}) \wedge match_{spec}(Q_{out}, T_{out}) \wedge match_{plugin}(Q,T)$$

We specialize this definition for the case of using Feature Terms as the Object Language, by using the *subsumption* ($\sqsubseteq$) relation as the basic matching relation.

$$match(Q,T) = T_{in} \sqsubseteq Q_{in} \wedge T_{out} \sqsubseteq Q_{out} \ \wedge T_{pre} \sqsubseteq Q_{pre} \wedge Q_{post} \sqsubseteq T_{post}$$

The idea of this matching relation is to assess whether a task is representative of a problem. A task is considered to be representative of a problem if, given an input of the type specified by the query, and assumed that the preconditions stated by the query holds, the solution to the task will provide all the information required by the query output signature, and guarantees that the query postconditions will hold afterwards.

The *task-selection* activity can be described as function ($\mathcal{Q} \times \mathcal{T} \rightarrow \mathcal{T}$) that retrieves from a library a set of tasks that are representative of the problem at hand, where representativeness is decided upon the verification of a query-task match. This function takes a query and the tasks in a library as input and return those tasks that verify a matching relation with the query. We can now define the result of the task-selection activity as a set $\mathcal{T}_Q$ of tasks satisfying a matching with Q, formally:

$$\mathcal{T}_Q = \{T_i \in \mathcal{T} | match(Q, T_i)\}$$

At the end of the Problem Specification process one task is selected by the user as the application task: $T_0 \in \mathcal{T}_\mathcal{Q}$. Otherwise, all the tasks satisfying (matching) the query can be used as legitimate starting points for the Knowledge Configuration process.

The final result of the Problem Specification is a collection of problem requirements and problem data. The resulting specification of problem requirements (Figure 4.18) includes an application task (the task to be configured), a specification of the kind of inputs provided to and outputs expected from the Problem Solving process, a collection of preconditions that are assumed to hold, a collection of postconditions to be satisfied, and a specification of domain-models to be used. It does not matter whether the task is straightforwardly selected by the user, by an automated service, or through an interactive problem-specification support tool, like the interactive broker presented in §4.4.4.

Problem requirements are specified by the following features (Figure 4.18):

- *Application task*: the task characterizing the type of problem to be solved. This task characterizes the type of problems to which the current problem is supposed to belong to.

Problem-Requirements
application-task → *String*
ontology → *Ontology*
inputs → set-of *Signature*
outputs → set-of *Signature*
preconditions → set-of *Formula*
postconditions → set-of *Formula*
domain-models → set-of *Domain-Model*
configuration-options → set-of *Configuration-Options*

Figure 4.18: Features used to specify problem requirements

- *Inputs* and *outputs* are specified as signature elements in the Object Language. These signatures characterize the kind of data provided as an input to the problem and the kind of data expected or required as a solution. These signatures verify the signature matching relation between of a query-task matching with respect to the application task T_0. More formally, $T_{in} \sqsubseteq Q_{in} \wedge T_{out} \sqsubseteq Q_{out}$.

- *Preconditions* are also used to select a task during the initial problem specification, but can be extended to better characterize the problem in order to refine the collection of candidate tasks. Finally, the preconditions included as problem requirements must guarantee that the preconditions of the application task hold, which is achieved by the condition $T_{pre} \sqsubseteq Q_{pre}$ applied during the task-selection activity.

- *Postconditions* are the most outstanding element of problem requirements, since they characterize the goals of the problem. Postconditions are used twice: once during the selection of the application task; and afterwards, during the Knowledge Configuration process, to verify whether a task-configuration is valid (satisfies the requirements). The application task must assure that the postconditions of the query will hold after solving that task, which is endorsed when all that is true for the postconditions of the task is also true for the postconditions of the query. This condition is represented by the conjunct $Q_{post} \sqsubseteq T_{post}$ in the specification of a query-task matching relation.

- *Domain-models* are used to restrict the domain knowledge that can be used by the selected capabilities during the Problem Solving process. Only domain-models included in the query are considered during the Knowledge Configuration process.

- *Configuration Options* are used to setup the Knowledge Configuration process. The main configuration option is the strategy to be used for the Knowledge Configuration process (§4.4.4). Three strategies have been implemented: one based on a deep-first search strategy, another one based

on a best-first strategy guided by past-configuration cases, and one more that is guided by the user though the user is provided with a heuristic help to decide which components are used. Other options control parameters that are specific of a particular configuration strategy.

The problem is specified using the same ontology used by the application task or there is an ontology mapping between the problem requirements ontology and the task ontology.

Figure 4.19 shows a screenshot of the problem specification interface as presented to the user when using the *Interactive Configuration* strategy (§4.4.4. This screen shows the user the collection of signature-elements and formulae that can be chosen by the user in order to specify the inputs, preconditions, postconditions characterizing his problem.

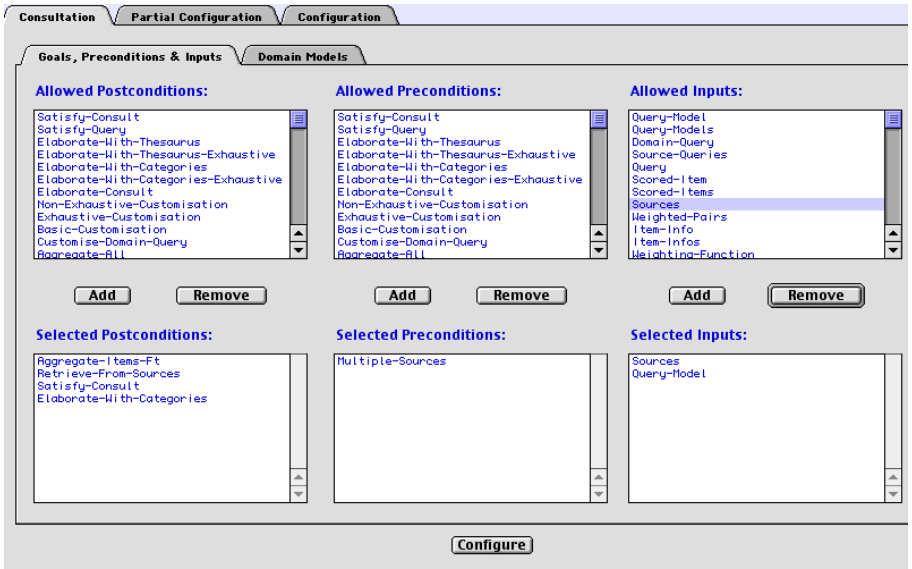


Figure 4.19: User Interface used to specify problem requirements.

Figure 4.20 shows a screenshots of the problem specification interface where the user can define the domain-models he is interested in.

4.4.3 Overview of the Knowledge Configuration process

The input for the Knowledge Configuration process is twofold: on the one hand it uses a collection of problem requirements, as described above, and on the other hand it uses a repository or *library* of component specifications to build up a configuration. The result of the Knowledge Configuration process is a *task-configuration* that, if complete and correct verifies the following: a) each task is bound at least to one capability that can achieve it, b) each capability requiring

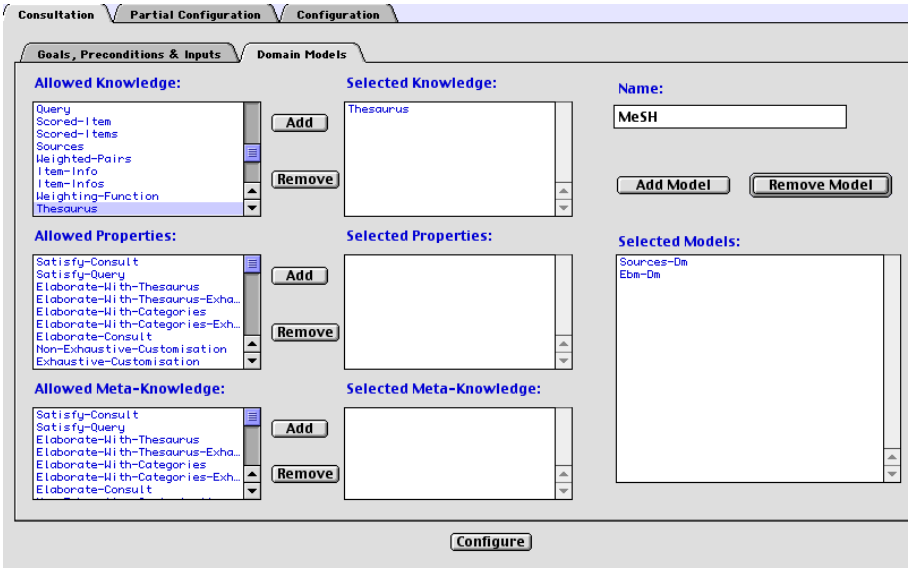


Figure 4.20: User Interface where the user defines the domain-models to be used during the Knowledge Configuration process.

knowledge is assigned a set of domain-models satisfying its assumptions, and c) the whole configuration complies to the problem requirements.

The Knowledge Configuration process has been designed and implemented as a search process in the space of partial configurations, where each state represents a partial configuration of a task.

The main information to be represented in a state is the set of task-capability *bindings* used in a partial configuration, but it also holds information about the requirements (inputs, preconditions and postconditions, domain-models and assumptions), those ones already satisfied and the ones yet to be satisfied.

The Knowledge Configuration process uses the problem requirements to generate an initial state. From the initial state, new states are generated by binding capabilities to tasks, where the bindings are realized through the verification of the matching relation between tasks and capabilities. The generation of new states continues until one of the new states is considered a final state; which occurs when the following conditions occur simultaneously: all tasks are bound to suitable capabilities, there are domain models satisfying the requirements of every capability included, and all the problem requirements are satisfied.

Figure 4.21 shows the main activities performed during the Knowledge Configuration process: *task-configuration*, *capability-configuration* and *verification*.

1. *Task-configuration*: The Knowledge Configuration process starts with an initial task and chooses a capability suitable for it (i.e. verifying a task-capability match).

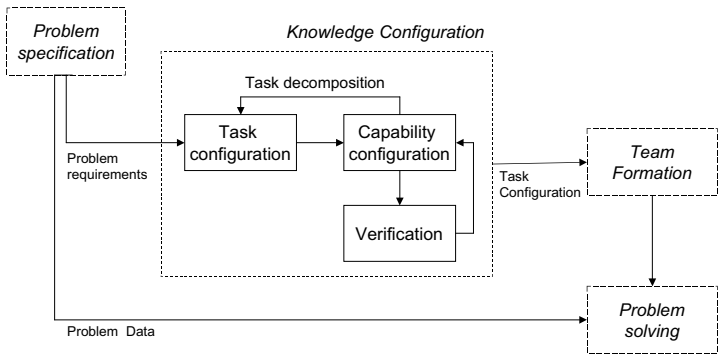


Figure 4.21: Main activities of the Knowledge Configuration process

2. *Capability-configuration*: The selected capability is configured by selecting domain-models compatible with it (verifying a capability-domain match). If a capability is a task-decomposer, then the task-configuration process should be performed for each subtask, which then becomes a recursive activity.

3. *Verification*: finally, verification is the process of checking whether the global problem requirements are met or not. If the global problem requirements are met —configuration is correct— and the configuration is complete, then the Knowledge Configuration ends, and the resulting task-configuration can be used to guide the Team Formation process. The point is that a verified task-configuration can be considered as design-level description of an application to be performed by a team of agents.

A task-configuration is recursively defined in term of a capability-configuration that boils down to a task-configuration for each subtask of the task-decomposer (a configuration schema), until all tasks are bound to a capability and thus there are no more subtasks to spawn from. The result is a tree of tasks that are bound to capabilities, and capabilities bound to domain-models, as shown in Figure 4.22.

4.4.4 Strategies for the Knowledge Configuration process

We have already introduced the notion of the Knowledge Configuration as a search process among the space of possible configurations: from an initial state the Knowledge Configuration process explores successor states according to some order until reaching a final state. We have implemented three different strategies for the search process, depending on the kind of user and the availability of past configuration cases, namely.

We differentiate between experts users, which are knowledgeable of the OR-CAS KMF and the Knowledge Configuration process (usually the knowledge

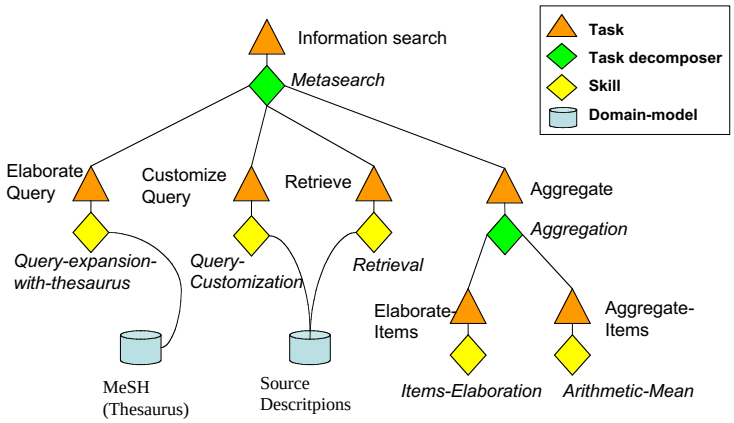


Figure 4.22: Example of a task-configuration

engineer), and the final user, which does not have such a knowledge.

The three strategies implemented for the Knowledge Configuration process are the following, namely: *Search and Subsumes*, *Constructive Adaption* and *Interactive Configuration*.

- *Search and Subsume* is appropriate for non expert users whether past configurations are not available or are not desired. The *Search and Subsume* configuration mode implements a depth first strategy for searching. This strategy uses subsumption for retrieving capabilities that match a task. For each retrieved capability, a new state (called successor state) is generated. The new states generated are added to a stack of open states; and the next state to be explored is always the head of this stack.

- *Constructive Adaption* is appropriate when the user (expert or not) wants to use *Case-Based Reasoning* (CBR) to drive the search process. The *Constructive Adaptation* strategy follows a *best-first* search process in the state space [Plaza and Arcos, 2002]. There is a heuristic function that assesses which state is "best", based on problem requirements previously used to configure a system (i.e. previous configurations here used as cases). This strategy uses past configurations to order the successor states according to a measure of similarity to the current state. A similarity measure called SHAUD [Armengol and Plaza, 2001] is used to evaluates the similarity between Feature Terms structures. The result is a ranking of past configuration cases that are used to order the states. Then, once we have reordered the states, the search process selects to expand the state with a higher heuristic value based on case-based similarity.

- *Interactive Configuration* is appropriate for expert users who want to have more control over the Knowledge Configuration process. This strategy in-

terleaves the specification and the configuration phases until a complete configuration is found. The *Interactive Configuration* strategy operates through a graphical user interface that shows the partial configuration, the available components and other information. Once the user selects a task to be configured, the interface presents the capabilities that are suitable for that task (those matching that task). These capabilities are ranked with the similarity measure used in the case-based strategy (Constructive Adaptation) in order to guide the user about which are the capabilities recommended by the system upon past configuration experience, but the user is free to select any capability based in its own criteria. If the selected capability is a task-decomposer, then the capability selection process is repeated for every subtask. In order to facilitate the user decision, the interface presents some extra information about the components and information about the current state of the configuration. The configuration service interleaves the problem specification and the task-configuration activities, and brings the user the possibility of deciding the next state to follow by selecting the capability to bind to the current task (from the set of suitable capabilities for that task). This mode is also useful for a knowledge engineer to build a initial case-base of configuration examples, thus allowing the end-user to use the constructive-adaptation strategy.

Figure 4.23 shows an example of a partial configuration as presented by the interface of the Interactive Configuration mode. The left part shows the current configuration of a task; and the rest of the interface shows from left to right and up to down the following information: first row shows available capabilities that are suitable for the task at hand, ranked according to the similarity of the current problem to past configuration cases, and a description of the capability currently selected by the user; the second row includes the pending preconditions, assumptions and goals (postconditions), and the unavailable or missing knowledge-roles; and finally, the bottom row of the interface shows the already achieved preconditions, postconditions, assumptions and knowledge-roles. In particular, this is an example from the WIM configurable application, which is described in Chapter 7. In this example, the Information-Search task is being configured. Only the configuration of the subtask Aggregate-Items remains unconcluded. The user has to select one of the four aggregation capabilities that are suitable for that task: the *Average*, the *Weighted-mean*, the *OWA* or the *WOWA*.

4.4.5 Searching the Configuration Space

The Knowledge Configuration process is approached as a search process over the space of possible configurations in order to find a configuration that is complete and satisfies all the requirements of the problem at hand.

The search space is $\mathcal{K}(\mathbb{L})$, the set of possible (partial and complete) configurations given a component library $\mathbb{L}$ and a query containing the requirements of the problem (Q). Moreover, a configuration $K \in \mathcal{K}$ can be a solution for the

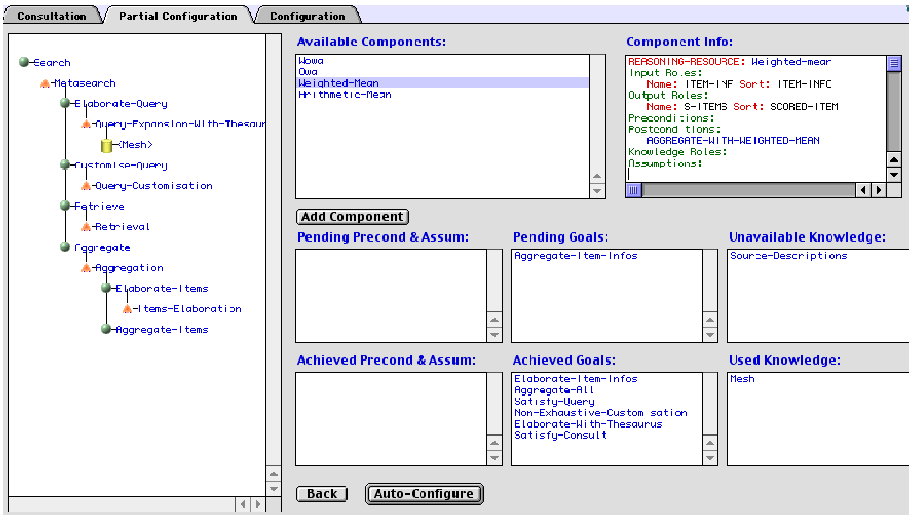


Figure 4.23: Interface that shows a partial task-configuration

query only if K is complete.

Let's start defining a query as a collection of problem requirements and optionally a selected application task.

Definition 4.11 ***(Query)*** *A query* $Q \in \mathcal{Q}$ *is a tuple* $Q = \langle T_0, in, out, pre, post, dm \rangle$

where T_0 is the application task, which characterizes the type of problem the application will be configured for, in, out are the input and output signatures describing the type of data available and the type of data expected as a result of the Cooperative Problem-Solving process, preconditions *pre* are properties that are stated to be true, postconditions *post* are properties to be achieved by the "execution" of the configuration (the performance of a CPS process based on that configuration), and $dm \subset \mathcal{M}$ is the set of allowed domain-models.

The Knowledge Configuration search process starts with a query Q and an empty configuration, and searches new states that model more detailed configurations by adding configuration schemas and recursively configuring them until a complete configuration is found. The *Constructive Adaptation* model tells us that we can improve the construction of the solution by using cases to decide which states to explore first—i.e. which configuration schemas to add to a given partial configuration. This search process adds configuration schemas until a complete configuration K is reached, and then checks if this K satisfies the query Q: if correct then a solution has been found and the process terminates; otherwise the search algorithm proceeds exploring other branches.

We turn now to consider how to represent a *state* in the Knowledge Configuration process. The main issue to be represented in a state is the set of

task-capability bindings ($T \doteq C$) in effect within a partial configuration. For this purpose the state needs not to represent the whole configuration K but only the subset of tasks included in a configuration $T_K \subset \mathcal{T}$ that are bound to a capability.

The second important issue for a state is determining which pre- and post-conditions of the query Q are satisfied by the components involved in a partial configuration—and which are not yet satisfied. Finally, we are interested in states that represent valid configurations, so states have to satisfy the task-capability matching relation for every task-capability binding ($T \doteq C$) in a state.

Definition 4.12 *(State) A* state *Z for a configuration K in a Knowledge Configuration process with a query Q is a tuple* $Z(K,Q) = \langle cpre, cpost, opre, opost, okr, ckr, ot, tc \rangle$

where *cpre* and *cpost* are the closed preconditions and postconditions respectively (those in query Q that are satisfied in Z), *opre* and *opost* are the open pre- and post conditions (those in query Q that are not satisfied in Z), *okr* and *ckr* are the open and closed knowledge-roles (those required by capabilities bound to a task), *tc* is a set of task-capability bindings ($T \doteq C$), and *ot* is the set of open tasks (those tasks in the configuration K that are not bound to any capability).

A state Z is *valid* iff $\forall (T \doteq C) \in Z_{tc} \Rightarrow match(T,C) = true$.

State
open-postconditions → set-of *Formula*
open-preconditions → set-of *Formula*
closed-postconditions → set-of *Formula*
closed-preconditions → set-of *Formula*
open-knowledge-roles → set-of *Signature*
closed-knowledge-roles → set-of *Signature*
open-tasks → set-of *Task*
tc-bindings → set-of *Task-Capability-binding*

Figure 4.24: Features characterizing a state

Figure 4.24 shows the features characterizing a *state*: *open-postconditions* and *open-preconditions* are those postconditions and preconditions not yet satisfied in the current state; *closed-postconditions* and *closed-preconditions* are the preconditions and postconditions satisfied by the partial configuration; *open-tasks* attribute holds the tasks that have no capability bound to them; *open-* and *closed-knowledge-roles* refer to the knowledge-roles to be filled or already filled respectively, by a domain-model (from those established by the query); and *tc-bindings* is a collection of task-capability bindings (i.e. pairs composed of a task bound to a capability ($T \doteq C$)).

Next we are going to explain how the search proceeds for the Knowledge Configuration process. For this purpose we are going to answer with the following questions:

1. how initial states are generated from the library of components (§ 4.4.5);
2. how succesor states are generated (§ 4.4.5);
3. how the state search is guided by case-based retrieval (only for the Constructive Adaptation strategy, described in § 4.5);
4. and how final states (solutions) are detected (§ 4.4.5).

Initializing the search process

The initialization process takes the specification of problem requirements or query Q and a library of components $\mathbb{L}$ as inputs and produces the initial state as output (initial-states: $\mathcal{Q} \times \mathbb{L} \rightarrow \mathcal{Z}$).

The first state is generate according to the application task T_0 specified in the query Q:

$$Z^0 = \langle \emptyset, \emptyset, Q_{pre}, Q_{post}, T_0, \emptyset \rangle$$

The application task T_0 is a starting point for the search process, thus this task becomes an open task, since there is no capability bound to it yet. The pre-conditions and post-conditions of the query Q become open pre- and post-conditions, and there are neither closed pre- and post-conditions, nor task-capability bindings.

If there is no application task selected, then all tasks satisfying the query $\mathcal{T}_Q = \{T_i \in \mathcal{T} | match(Q, T_i)$ are established instead as legitimate starting points for the search process. Consequently, for each task $T_Q^i \subseteq \mathcal{T}_\mathcal{Q}$ an initial state is generated as follows:

$$Z^i = \langle \emptyset, \emptyset, Q_{pre}, Q_{post}, T_Q^i, \emptyset \rangle$$

Successor states

The process of generating *successor* states from a given state (the successors function) is basically the addition of a new task-capability binding to those present in the *tc-bindings* of the given state (Z_{tc}). We are interested in retrieving from the library a capability that matches one of the open tasks (Z_{ot}).

We take, for a state Z, a task from open tasks $T \in Z_{ot}$ and retrieve[2] a collection $\mathcal{C}_T$ of capabilities such that they match with T. In addition, only capabilities which knowledge-roles can be filled in by domain-models included

[2]The particular retrieval method (based on subsumption) that we use has been described in [Arcos and López de Mántaras, 1997].

in the query Q_{dm}, and which assumptions are guaranteed for all its knowledge-roles, are allowed. Therefore, we can now define the set of capabilities $\mathcal{C}_T$ that can be bound to a task, as follows:

$$\mathcal{C}_T = \{\ C \in \mathcal{C} \mid match(T, C) \wedge match(C, Q_{dm})\}$$

where $match(T, C)$ is a task-capability match (definitions 4.1 and 4.4) and $match(C, Q_{dm})$) is a capability-domain match (definitions 4.2 and 4.5), and Q_{dm} is the set of domain-models specified in the query Q.

A new successor state of Z is generated for each capability $C^h \in \mathcal{C}_T$. A successor state Z^h has no longer T as an open task and has a new binding $(T \doteq C^h) \in \mathcal{C}_T$. In addition, incorporating a new capability C^h achieves some new postconditions that were not yet achieved in Z; therefore the generation of the successor state updates the pre- and post-conditions that are open and closed.

When C^h is a task decomposer, it will introduce new subtasks that are to be considered now as open tasks—and, since a new open task can introduce new pre- and postconditions, the open and closed pre- and postconditions have to be revised accordingly. Thus, the successor state is

$$succ(Z, C^h) = \langle cpre^h, cpost^h, opre^h, opost^h, Z_{ot} \cup C^h_{st}, Z_{tp} \cup (T \doteq C^h)\rangle$$

and $cpre^h$, $cpost^h$, $opre^h$, $opost^h$ are the task-decomposer open and closed pre- and postconditions.

Another source of variability depends on the domain-models that can be sensibly used by the capability C^h. Let us consider the set of domain-models in the query $\mathcal{M}^h \subseteq Q_{dm}$ that satisfy the signature specification of the capability knowledge-roles C^h_{kr}. If there is a one to one mapping from the domain-models in $\mathcal{M}^h$ to the signatures in C^h_{kr} then the pair $(C^h, \mathcal{M}^h)$ is unique. However, if there is a many to one (non-injective) mapping from the domain-models in $\mathcal{M}^h$ to the signatures in C^h_{kr} then there may be several pairs $(C^h, \mathcal{M}^h_i)$ where $\mathcal{M}^h_i \subset \mathcal{M}^h$ has a one to one mapping to the signatures in C^h. In this situation one successor state is generated for each pair.

Final states

The verification of whether a state Z^G is a solution to the configuration problem (the `goal-test` function) has to test whether a (task) configuration is *complete* and *valid*:

1. A configuration is *complete* if all tasks are bound to some capability, thus all we need to check is whether there are no open tasks, i.e. whether $Z^G_{ot} = \emptyset$.

2. A configuration is *valid* if all the problem-requirements are satisfied, that is verified when there are no open-postconditions in the state, i.e. whether $Z_{opost} = \emptyset$

A more formal definition of a valid configuration is based on the verification of a satisfiability relation between the configuration obtained in state Z^G and the problem requirements imposed by the query Q ($sat(Q, Z^G)$) as follows:

$$sat(Q, Z^G) \Longleftrightarrow (Q_{pre} \Rightarrow Z^G_{pre}) \wedge (Z^G_{post} \Rightarrow Q_{post})$$

The same definition specialized with the subsumption relation over Feature Terms is the following:

$$sat(Q, Z^G) \Longleftrightarrow (Q_{post} \sqsubseteq Z^G_{post}) \wedge (Z^G_{pre} \sqsubseteq Q_{pre})$$

That is to say, the state Z^G satisfies the initial query Q when all postconditions imposed by the query are satisfied (subsumed) by the postconditions of state Z^G and when all preconditions required by the capabilities included in the configuration of the state Z^G are satisfied (subsumed) by the preconditions established by the query.

We know, by construction, that the completeness condition is assured by $Z^G_{ot} = \emptyset$ and the validness condition is assured by $Z_{opost} = \emptyset$.

4.5 Case-based Knowledge Configuration

The *Constructive Adaptation* strategy views the Knowledge Configuration process as a search process guided by case-base information.

The idea of using a case-based retrieval approach to rank the successor states according to the similarity of the current problem to past configuration problems. This is the reason to introduce the notion of a *configuration case* as a pair composed by a query and a past solution; specifically a configuration case is a pair (Q, K) where Q is a query containing problem requirements and K is a configuration of components (a task configuration) that satisfies Q.

Definition 4.13 ***(Configuration Case)*** *A configuration case is a pair (Q, K) where $Q \in \mathcal{Q}$ is a query containing problem requirements, and $K \in \mathcal{K}$ is a complete and valid configuration.*

A case base B is a collection of configuration cases $B = \{(Q, K)\}_{1...N}$.

Constructive Adaptation uses information of similar cases to guide the search process. Since in the ORCAS Knowledge Configuration process the main step is to choose a new capability hypothesis to include in a successor state, the retrieved cases are used to decide the selection of capability hypothesis. Let us suppose that we are trying to find the configuration for a query Q, that $CB = \{(Q_i, K_i)\}$ is a case base (a collection of cases), and that $Sim(Q, Q_i)$ is a similarity measure that assesses the relevance of cases in CB with respect to the problem query Q.

Since a new successor state of Z is generated for each capability in $\mathcal{C}_T$ let us call $\mathcal{Z}_T$ the set of those states. Moreover, at any point in the search process there is a set of states that are "open", i.e. states from which successor states have yet to be generated; let us call $\mathcal{Z}_{open}$ the set of those states. We will use similar

cases to order the set of all open states $\mathcal{Z}_T \cup \mathcal{Z}_{open}$; the search process will use this ordering to decide the next state from which successor states are spawned. Therefore, the search process follows a best-search strategy where a similarity assessment of states against past configuration cases is used as an heuristics to decide the "best" successor state.

We use the similarity measure $Sim(Q, Q_i)$ to rank the cases in the case base $CB = \{(Q_i, K_i)\}$ with respect to their similarity to our current problem Q. Once they are ordered, we need to transfer this ordering to the the set of all open states $\mathcal{Z}_T \cup \mathcal{Z}_{open}$. For this purpose, let us define two new elements:

- $\mathcal{C}_{K_i}$ is the set of capabilities used in some task on configuration K_i.
- C_{Z_j} is the new capability that has been introduced as hypothesis when state Z_j was generated.

Now, the new ordering over states is computed by transferring similarity S over cases to a similarity S_Z over states. Specifically, we define state similarity $S_Z(Q, Z_j)$ as follows:

$$S_Z(Q, Z_j) = \max_{Q_i \in CB} \{Sim(Q, Q_i) | C_{Z_j} \in \mathcal{C}_{K_i}\}$$

That is to say, for each open state $Z_j \in \mathcal{Z}_T \cup \mathcal{Z}_{open}$ we consider the newly added hypothesis (C_{Z_j}), then we check in which cases the capability C_{Z_j} appears as part of the configuration ($\mathcal{C}_{K_i}$), and we take the most similar (to Q) as the degree of similarity for state Z_j.

Thus, Constructive Adaptation proceeds by constructing the solution guided by the information of cases embodied by the similarity measure S_Z. At every decision step, Constructive Adaptation selects the state Z_{max} w.r.t S_Z as the best node to expand the search tree. Or, in other words, the best-first process uses S_Z in the cost function that decides whether a state Z is better than another state Z', as follows

$$\textit{Similarity-fn}(Z, Z') = S_Z(Q, Z) > S_Z(Q, Z')$$

.

Figure 4.25 shows the relation between the problem space and the solution space according to a similarity (distance) assessment. In ORCAS the problem space is defined by the set of possible problem requirements, while the solution space is the set of possible configurations. Given a problem that is similar to the current problem (R distance), then the solution of the previous case will be near to the solution for the current problem (A distance).

4.6 Configuration as reuse

One of the motivations of this work can be summed up in the idea of *reuse*, which is closely related to the notion of *domain independence*. We aim at providing a

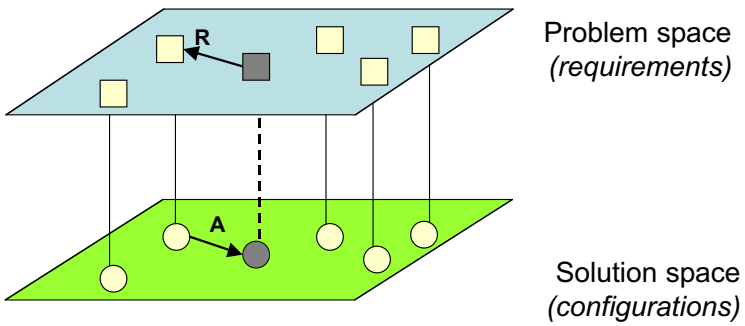


Figure 4.25: Relation between similarities in the problem space and the solution space

framework that maximizes the reuse of agent capabilities over new application domains. But what we mean by "reuse"?

Looking at the literature we have realized that there are two main approaches to the reuse issue, though they can adopt different nomenclatures and may be representative of different research lines, yet we can classify them in two general categories, that we call *engineering-oriented reuse* and *execution-oriented reuse*.

- *Engineering-oriented reuse*: this category includes those efforts devoted to the design of new applications by retrieving and adapting existing components. In general, we consider this class of reuse when components cannot be automatically composed and connected, but they rather require some adaptation in order to be executed. Most of the work carried over in the field of Knowledge Engineering and CBSD fall within this category. In general, there are very general components (like the so called "Off-the-Shelf" components) that can be parameterized to fit the requirements of a new problem, or it is necessary to modify the code of some components and then recompile. Usually there is a single application that is configured of tailored for a specific application by a software or knowledge engineer. This approach allows to develop very specific applications with reduced development costs, and enables the interoperation of fairly heterogeneous components, but requires the participation of an expert (the knowledge or software engineer).

- *Execution-oriented reuse*: this category includes those systems where components can be connected and executed in a very dynamic way, with minimal human guidance. The point is that the final user should be able to drive the reuse of existing components without a deep knowledge of the components. The notions of "on-the-fly or "plug-and-play" applications fall within this category. The idea is that there exist a library of components that can be straightforward connected to interact, without

adaptation or modification of the code. Therefore, there are many possible configurations that can be configured on-demand to better fit the problem at hand, rather than configuring a single application for a very specific problem.

The ORCAS framework maximizes the reuse of existing components in both the above approaches to reuse. The key to maximize reuse is the decoupling of the different types of components in the Abstract Architecture, even at the semantic level, by allowing them to be specified using independent ontologies providing the *semantics* of the components.

However, the Knowledge Configuration process as discussed here falls within the second approach to reuse (Execution-oriented reuse), since it is an automated process that allows a non expert user to specify the requirements of the problem at hand, and is able to obtain a configuration of components that can be then operationalized on runtime by forming a team of agents customized for that configuration. We assume that all the components share the same ontologies and use the same infrastructure to communicate; otherwise, it cannot be guaranteed that a configuration is found, or that two agents selected for a team can interact. Further work can be started here to work upon the consideration of different ontologies and the application of ontology-mappings to avoid ontology mismatches.

The engineering aspects concerning the use of the ORCAS KMF (e.g. the use of ontology mappings) are out of the scope of this work, which is focused on the automatic configuration of MAS based applications.

Concerning the issues of reuse and configurable applications, we give now some definitions to better characterize the idea of reuse as used in the Knowledge Configuration process. First of all we introduce the notion of *library* and *on-the-fly application*, and then the notion of a *configurable application*.

An *On-the-fly Application* is a particular configuration of problem-solving components (tasks and capabilities) that are able to solve a task T_0 in some application domain, according to some problem requirements. An on-the-fly application is the result of connecting some tasks and capabilities of a particular task-configuration to the domain-models characterizing an specific application domain.

Definition 4.14 ***(On-the-fly Application)*** *An on-the-fly application* $\mathbf{A}$ *is a tuple* $\mathbf{A}(Q, \mathbb{L}) = \langle \mathrm{T}, \mathrm{C}, \mathrm{M}, K, \mathbb{O} \rangle$

where $\mathbb{L}$ is a library of domain-independent problem-solving components (tasks and capabilities, represented as $\mathcal{T}$ and $\mathcal{C}$) Q is a query containing problem requirements, including an application task $T_0 \in \mathcal{T}$, and a collection of domain-model characterizing the application domain Q_{dm}; K is a task-configuration for the application task T_0; $\mathrm{T} \subseteq \mathcal{T}$ is the set of tasks used in K, $\mathrm{C} \subseteq \mathcal{C}$ is a set of capabilities included in K; M is a set of domain-models to be used by the capabilities in C; and $\mathbb{O}$ is the Object Language.

A *Configurable application* is the result of linking a library of components (tasks and capabilities) to a collection of domain-models characterizing some application domain. In a configurable application there are multiple possible configurations over the components of the library. While an on-the-fly application is a particular configuration of components to solve a specific task, a configurable application is potentially able to solve many types of problems (each task defines a problem type), and the same type of problems can be solved in different ways, using different task-configurations. A configurable application have multiple, alternative capabilities to solve the same class of problems, or there are multiple domain-models available to choose from.

Definition 4.15 ***(Configurable Application)*** *A configurable application is a tuple* $\mathbf{A}(\mathbb{L}, \mathcal{M}) = \langle \mathcal{T}, \mathcal{C}, \mathcal{M}, \mathcal{K}, \mathbb{O} \rangle$

where $\mathbb{L}$ is a library of problem-solving components (tasks and capabilities), $\mathcal{M}$ is a collection of domain-models characterizing the application domain, $\mathcal{T}$ is the set of tasks in $\mathbb{L}$, $\mathcal{C}$ is the set of capabilities in $\mathbb{L}$, $\mathcal{M}$ is a set of domain-models to be used by $\mathcal{C}$ (all the components share a common ontology or there exist a collection of mappings between ontologies such that they be treated as sharing a single ontology), and there exist a collection of task-configurations over the components $\mathcal{K}$ ($\mathcal{K} \in \mathbb{K}$). A configurable application is not an application in the classical sense, it is rather a collection of components that can be connected and configured "on-the-fly", by finding one of the possible configurations in $\mathcal{K}$ that satisfies the requirements of a specific problem, as far as the problem can be characterized by one the tasks in $\mathcal{T}$. In other words, a configurable application can be seen as a collection of potential on-the-fly applications sharing the same application domain.

An example of a configurable application is shown in Chapter **??**, WIM.

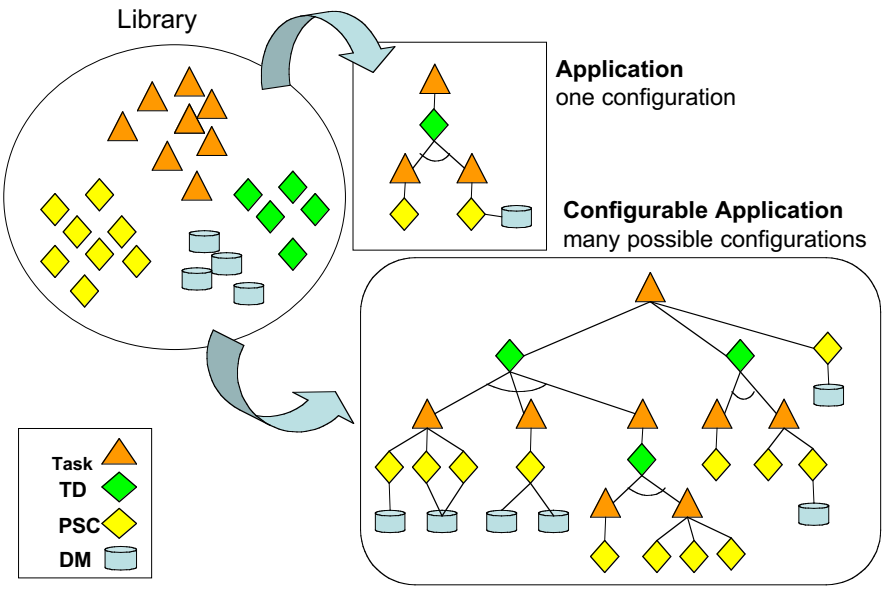


Figure 4.26: Library, application and configurable application

Chapter 5

The Operational Framework

This chapter describes an operational model of the Cooperative Problem-Solving process for Multi-Agent Systems, and which is the role played by the Knowledge-Modelling Framework within this process.

5.1 Introduction

The Operational Framework describes a mapping from the concepts in the Knowledge-Modelling Framework to concepts from Multi-Agent Systems and Cooperative Problem Solving. Specifically, the Operational Framework describes how a task-configuration —obtained by the Knowledge Configuration process— can be operationalized by forming a customized team of problem solving agents on-demand, according to stated problem requirements. Moreover, the Operational Framework describes also the communication and the coordination mechanisms required by agents to carry over the Cooperative Problem-Solving process according to a task-configuration.

The Operational Framework extends the Knowledge Modelling Framework to develop a full-fledged Agent Capability Description Language (ACDL). The ORCAS ACDL is used in open Multi-Agent Systems by *requesters* willing to solve a problem, *providers* of problem solving capabilities, and *middle agents* responsible for mediating between them (e.g. brokers and matchmakers).

1. *Requesters* use the ACDL to put a query Q describing the type of problem to be solved, characterized by a task (the application task T_0), a collection of domain models characterizing the application domain (Q_{dm}), and other requirements of the problem (preconditions and postconditions).

2. *Providers* use the ACDL to describe the tasks they can solved and the capabilities they are equipped with.

3. *Middle agents* are used in open MAS to mediate between requesters and providers. Middle agents are responsible for locating appropriate providers

for a given request and facilitating the interaction between requester and providers. For instance, a broker frees the requester of knowing the details required to invoke and interact with each specific provider, so the requester only have to know how to interact with the middle agent and not with each potential provider.

In our approach to the Cooperative Problem Solving (CPS) process and also in our particular implementation of an open agent platform supporting the CPS process, the ACDL is used for the following activities: (1) in the automatic design of agent teams at the knowledge level, as a configuration of components (tasks, agent capabilities and domain models) satisfying stated problem requirements; (2) to guide the team formation process according to the configuration of components at the knowledge-level; and (3) to coordinate the behavior of team members during the teamwork.

This chapter is divided as follows: we start with an overview of the Cooperative Problem-Solving (CPS) process in §5.2; the extensions of the ORCAS KMF to become an Agent Capability Description Language (ACDL) are described in §5.4; the main concepts of the ORCAS team model are defined in §5.3; the Team Formation process is described in §5.5; the ORCAS model of Teamwork is presented next, in §5.6; and the Chapter ends with a brief discussion of some extensions of the CPS process in §5.7 .

5.2 The Cooperative Problem-Solving process

The view on Multi-Agent Systems as decoupled networks of autonomous entities is usually associated to a distributed model of expertise: A MAS is regarded as a collection of specialized agents with complementary skills.

Most of the research done in the field of *Cooperative Problem Solving* (CPS) (e.g. the models based in the Contract Net protocol [Smith, 1940] or derived from the Generalized Global Planning approach [Durfee, 1988]) falls into one or more or the stages of a general model of the Cooperative Problem-Solving process as presented in [Wooldridge and Jennings, 1994], which consists of four stages: *recognition*, *team formation*, *planning* and *execution*.

1. *Recognition*: the CPS begins when some agent recognizes the potential for cooperative action. This recognition may come about because an agent has a goal that it does not have either the ability or the resources to achieve on its own, or else because the agent prefers a cooperative solution in expectation of getting some benefit.

2. *Team formation*: during this stage, the agent that recognized the potential for the cooperative action at stage (1) requests assistance. If this stage is successful, it will end with a group of agents having some kind of nominal commitment to collective action.

3. *Plan negotiation*: during this stage, the agents proceed to negotiate a joint plan which they believe will achieve the desired goal.

4. *Execution (Team action)*: During this stage, the newly agreed plan of joint action is executed by the agents, which maintain a close-knit relationship throughout. This relationship is defined by a convention, which every agent follows.

Actually, these four stages are *iterative*, in that if one stage fails, agents may return to previous stages. Although the proposers of this model believe that most instances of CPS exhibit these stages in some form (either explicitly or implicitly), they stressed that the model is idealized. In other words, there are cases that the model cannot account for [Wooldridge and Jennings, 1999].

In concordance with the proposers of this model, we think it is well suited for a number of situations, but it is not adequate for others (the reader is referred to [Wooldridge and Jennings, 1994, Wooldridge and Jennings, 1999] for a deeper understanding of the model). Since team formation is not guided by a preplan to achieve the overall goal, but is just a commitment to joint action, neither the agents joining a team (committing to carry on joint action) are guaranteed to play one role in the team once a plan was decided at the subsequent planning stage, nor the resulting team assures that a global plan can be found.

This uncertainty may be not a problem if the MAS is composed of quite homogeneous problem solving agents with a common range of skills. In such homogeneous (in term of functionalities) agent societies, the very same agent could potentially occupy many different positions within a team, and thus the possibility of forming a successful team grows up. This approach has been adopted mostly when cooperation is defined as acting with others for a common purpose and a common benefit [Norman, 1994] where the purpose should be motivated by an intention to act together —a joint intention [Cohen and Levesque, 1990, Levesque, 1990, Cohen and Levesque, 1991], and resolved by a commitment to joint activity [Bratman, 1992, Jennings, 1993].

This class of cooperation relying on motivational attitudes is sometimes called collaboration [Wilsker, 1996, Grosz and Kraus, 1996] to differentiate it from other classes of cooperation. It is not surprising then that implemented frameworks inspired by a collaborative approach to cooperation are usually applied in scenarios where the team-oriented agents are homogeneous and their roles typically represent either authority relations, such as military rank, or high-level capability descriptions [Tambe, 1997, Pynadath et al., 1999].

Moreover, if the MAS is composed of specialized agents equipped with very specific capabilities, the following bizarre situations may happen: first, agents that have joined the team may not be needed after the plan negotiation have finished; second, team roles may remain still unassigned to agents after negotiating the plan; and third, the plan may remain incomplete. In order to prevent these types of failure (specially the second and third cases, which suppose the team is not able to achieve the global goal at all), specialized agents should be able to reason about goals and plans in order to acquire team goals, to identify

the roles that contribute to the fulfillment of a team goal, and to decide whether to commit to a particular team role. Team formation and also plan negotiation without a initial guide on the types of tasks to be solved and the capabilities required, can provoke an exponential grow in the number of teams and plans to be considered, and a blow-out in communication overload as the size of the population grows up or the complexity of the team goal increases.

Other researchers have explored the utility of using an initial plan to guide the team formation process. The SharedPlans theory [Grosz and Kraus, 1996] and the frameworks based on it, e.g. [Giampapa and Sycara, 2002], have emphasized the need for a common, high-level team model that allows agents to understand all requirements for plans that might achieve a team goal. Team plans are used by agents to acquire goals, to identify roles and to relate individual goals to team goals. An initiator of a cooperative activity can use an initial team plan to know the functionalities or competencies required to achieve the overall goals.

An initial plan allows the initiator of the team formation process to know which are the subgoals and (optionally) the actions or capabilities required to achieve each subgoal. Therefore, the initiator can use the initial plan to guide the team formation process [Tidhar et al., 1996]. An initial team plan allows the initiator of the cooperative activity to contact only with the agents holding the required capabilities, thus increasing the possibility of success, and reducing the complexity of both the team formation and the plan negotiation processes.

Another issue concerning the CPS process and related with the idea of guided team formation is that of problem requirements: how to design a team according to the requirements of a specific problem, rather than selecting a plan according to a fixed task. The CPS model in [Wooldridge and Jennings, 1999] does not explicitly address the utility of constraining the competence or behavior of a team to satisfy stated problem requirements or comply to user preferences. Recent initiatives in MAS planning have introduced case-based [Munoz-Avila et al., 1999] and conversational planners to build the initial plans to be adopted by a team [Giampapa and Sycara, 2001].

Moreover, that model of the CPS process devises an internal perspective on agents, in which the agent's internal state is used as the basis for evaluation. Concerning this issue, though we recognize there are well founded reasons to adopt an internal perspective, we think a external view has also some advantages and, in particular, it is more appropriate for open systems because it avoids imposing a model of the agent architecture to external agent developers. We are developing our model under this assumption, and thus we try to impose minimal requirements on the internals of agents willing to participate in a CPS process beyond the use of the ORCAS Knowledge Modelling Ontology. These requirements consist basically of a shared communication language and a set of interaction protocols enabling team action. Therefore, we avoid imposing neither a specific agent architecture, nor a model of cooperation based on mental attitudes. Instead, an external view centered on the observable patterns of agent communication and commitment is adopted here.

Summarizing, there are some issues related to Cooperative MAS not covered

by the general model of the CPS process in [Wooldridge and Jennings, 1999]: (1) the generation of an initial plan to guide the team formation process; (2) the consideration of user preferences and specific problem requirements to constrain the composition of a team; and (3) the use of a external view centered on observable events like the illocutionary acts rather than an internal view imposing a particular agent architecture.

As a result of our work upon these issues we have conceived a new model of the CPS process that is based on the use of a Knowledge-Modelling Framework to describe a MAS at an abstract-level, and introduces a Knowledge-Configuration process as a mechanism to design the behavior of an agent team by building an initial team "plan" satisfying stated problem requirements. Such initial plans can be used to drive the team formation process and to coordinate agent behavior during the teamwork.

Now we are going to carry out a more detailed review of the main activities involved in the CPS process —team formation, planning and execution— in order to compare our approach to the planning based approaches that are commonly used in cooperative MAS.

Team formation is defined as the process of selecting a group of agents that have complimentary skills to achieve a given goal [Tidhar et al., 1996]. Typically, team formation has been divided in two activities: selecting a group of agents that will attempt to achieve the team goal, and selecting a combination of actions that agents must perform to achieve the goal [Levesque, 1990, Cohen and Levesque, 1991, Rao and Georgeff, 1995, Grosz and Kraus, 1996, Tambe, 1997], also approached as a plan negotiation process [Wooldridge and Jennings, 1999]. This combination of actions is typically described as a sequence of actions or a *plan* [Georgeff and Lansky, 1987, Bratman, 1988, Rao et al., 1992, Grosz and Kraus, 1993, Sonenberg et al., 1994, Grosz et al., 1999, Tate, 1998]. In many approaches there are partial plans hold by different agents that must negotiate a given plan until consensus is reached about an agreed global plan [Ephrati and Rosenschein, 1996]. In other approaches the plan is built by merging individual plans —like the approaches based on the SharedPlans theory, for instance in the RETSINA teamwork model [Giampapa and Sycara, 2002], — until all the tasks required by the global plan are assigned. While some approaches start the selection of team members without an initial plan, other approaches [Tidhar et al., 1996, Giampapa and Sycara, 2001] use it to drive the team formation process, even when the planning process is distributed among agents [Clement and Durfee, 1999].

One of the preferred approaches to represent plans in MAS is based on using some kind of hierarchical plans, like Hierarchical Task Networks (HTN) [Erol et al., 1994, Erol, 1995]. The way in which an HTN structure decomposes a task into subtasks is similar to the way tasks are decomposed by a task-decomposer in the ORCAS KMF. Nevertheless, ORCAS configuration structures are not oriented towards planning algorithms; instead, ORCAS structures are designed to maximize the reuse of agent capabilities by decoupling the descrip-

tion of capabilities from the application domain through the use of independent ontologies to describe them both. In spite of these differences, ORCAS configuration structures are used as recipes about the tasks (or goals) to achieve and the capabilities (or actions) required to achieve them, and thus they play the same role than a plan.

In ORCAS there are three types of structure concerning planning: task-decomposition schemas (subtasks introduced by a task-decomposer and ordered by the operational description); *configuration schemas* resulting of binding a task-decomposition schema to a collection of capabilities (one capability is bound to each subtask); and *task-configurations*, which are composed of interrelated configuration schemas (see §4.4.1).

An approach to multi-agent planning is that of obtaining a global plan by merging individual, (usually partial) plans. In ORCAS the individual plans are partial (agents have a local view), and are represented by task-decomposers and configuration-schemas, which are two ways of representing how to achieve a task by decomposing it into subtasks. A configuration-schema is a more specific representation than a task-decomposer, since the former includes the capabilities required to solve each task, while a task-decomposer informs only about the subtasks of a decomposition, but not about the capabilities required to achieve them.

The role of a global team plan in ORCAS is played by a *task-configuration*, since a task-configuration is used as a recipe about the actions (the capabilities) required from team members to achieve the goals of the team (represented in ORCAS by an application task plus some extra problem requirements).

Task-configurations obtained at the Knowledge Configuration process are used to guide the team formation process and to coordinate team members during the Teamwork process: an agent willing to start a cooperative activity uses such an initial plan to know which are the tasks to be solved, which are the capabilities to apply, and which is the knowledge to be used by the selected capabilities. The agent responsible for coordinating the team formation process can use the information provided by a task-configuration to select the agents that are potential candidates to join the team (though a yellow pages service): only agents with the required capabilities are considered, thus the number of possible teams to be considered is reduced and the communication requirements decrease. In addition, the Knowledge-Configuration process ensures that a task-configuration satisfies stated problem requirements, therefore the teams that are formed complying to a task-configuration are guaranteed to satisfy the requirements of the problem too.

Teamwork is the process carried out by a team of agents in order to achieve a global team goal. In ORCAS the team goal is initially represented by the application task and subsequently refined by a task-configuration. A task-configuration contains a specification of tasks and subtasks to be achieved (a hierarchical task decomposition structure), the competence required to solve those tasks (the capabilities), and the specification of the application domain (the domain-models). The global behavior of a team during the Teamwork process is guided by the

task-configuration, since team members commit to solve the application task by applying the capabilities and domain-models specified in the task-configuration.

To move from the knowledge level to the operational level, we have to match concepts from a task-configuration to agent concepts. Our proposal is to define a one-to-one mapping between tasks in a task-configuration and roles played within a team, that we call *team-roles*. There is one team-role for each task, where each team-role contains the information an agent needs to solve a task in the context of a team. During the execution stage of the Cooperative Problem-Solving process, team members have to cooperate with other team mates in order to achieve the overall goal. Specifically, we are interested here in a hierarchical, top-down style of cooperation, since this style of cooperation fits well the hierarchical structure of a task-configuration.

A motivation for separating the Knowledge Configuration process from the Team Formation process is to exploit the fact that the specification of agent capabilities remains stable throughout long periods of time, whereas there are dynamic aspects of the system or the environment that change very quickly and are not deterministic (i.e. workload, network traffic, system failures). The Knowledge Configuration process aims to explore the utility of a configuration of capabilities in terms of their static and abstract descriptions, and keeping those descriptions aside from the the dynamic and operational aspects involved in the CPS process (e.g. the workload of a problem solving agent), which are managed during the Team Formation and Teamwork processes.

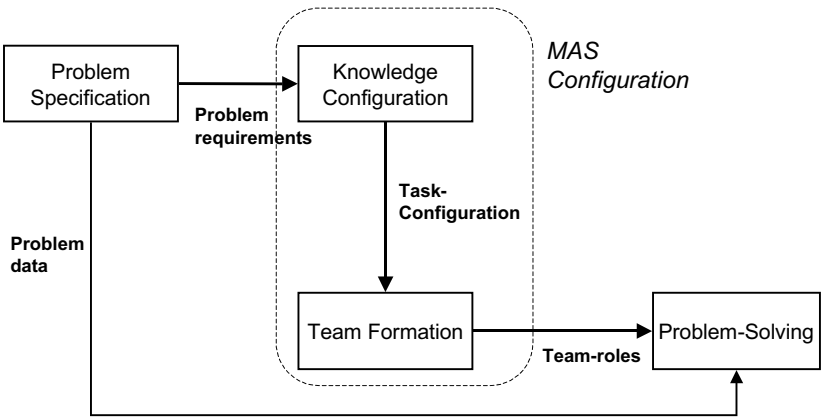


Figure 5.1: The ORCAS model of the Cooperative Problem-Solving process

Figure 5.1 shows the main elements of the Cooperative Problem Solving process and the main relations between them. The CPS process starts with a Problem Specification process in which the problem requirements and the problem data are supplied. The Knowledge Configuration process takes the problem requirements and a library of components as input and builds a task-configuration.

The Team Formation process uses a task-configuration to form a new team of agents satisfying the conditions established by that task-configuration (i.e. agents commit to the conditions enforced on the tasks allocated to it, its team roles). The outcome of the Team Formation process is a configuration of the team expressed as a collection of interrelated team roles to be played by the selected team members, and a group of specific agents assigned to these roles. Finally, during the Teamwork process the new team solves the problem at hand according to the task-configuration obtained at the Knowledge Configuration process and a specification of the input data provided at the Problem Specification process.

Although the CPS process as showed in Figure 5.1 seems to follow a sequential control flow, this model is just a simplification that is intended to highlight the cornerstones of the model. Once the different subprocesses of the CPS process have been explained we are in a better position to extend the model in order to deal with more complex situations. Actually, we have added mechanisms for interleaving all the stages of the Cooperative Problem-Solving process. This feature will allow a team to be reconfigured in order to deal with dynamic conditions and events encountered during the different stages of the CPS process. Some extensions of this model are described in § refsec:extensions.

5.3 Team model

The ORCAS team model is defined upon concepts from the Knowledge Modelling Framework, specifically, ORCAS teams are defined according to an abstract model of teamwork based on the structure and the meaning of a task-configuration.

A *team* is a group of agents that commits to solve a problem in a cooperative way, according to a task-configuration. A *team* is composed of agents that have complimentary capabilities to achieve a global goal and are assigned to different *roles* within the team. A team has the goal of solving a specific problem by using a task-configuration as a recipe about the task to be solved (specifying the global goal), a decomposition of the main task into subtasks, the capabilities to be applied, and the domain knowledge required.

As explained in §4.4, the result of the Knowledge Configuration process is a *task-configuration*, a hierarchical structure where nodes are triplets consisting of a task, a capability bound to the task, and optionally some domain-model satisfying the assumptions of the capability. In order to understand the way a task-configuration is operationalized by a team of agents, it is necessary to establish a mapping between the concepts involved in a task-configuration and concepts from Multi-Agent Systems.

A task-configuration in the ORCAS team model plays the role of a team plan, since it is used as a recipe of actions to achieve a global goal. The configured task represents a global team goal and a plan to achieve that goal. Specifically, tasks represent goals and subgoals; skills represent primitive, non decompos-

able actions to achieve goals; and task decomposers play the role of sub-plans in decomposing a task (a goal) into subtasks (subgoals). Multiple capabilities allowed for the same task represent alternative ways of solving a problem, or alternative ways of decomposing a problem into subproblems. In addition, a task-configuration contain the domain models satisfying the knowledge requirements of the selected capabilities.

In order to map elements from the KMF to elements of agent-based teamwork, we introduce the notion of *team-role*. A team-role defines the functions assigned to a position within the team. We establish a one-to-one correspondence between tasks and team-roles: there is a team-role for each task within a task-configuration. A task-configuration follows a hierarchical task decomposition structure, and a team in ORCAS is based on a task-configuration; consequently, teams are also organized hierarchically, as a nested structure of teams and subteams (Figure 5.2).

Team-roles define the competence required by the different members of a team in order to achieve a team goal. A team-role includes all the information required to solve one of the tasks of a task-configuration, that is to say, a task, a capability bound to it, and optionally a set of domain models required by the selected capability. When the capability bound to a task is a task-decomposer, a team role is defined for the task being decomposed, and a new team-role is created for each subtask. The agent playing the role of the task-decomposer acts as the *coordinator* (supervisor or leader) of the team-roles assigned to each subtask, that are in some sense "subordinated" to the coordinator, though the precise nature of the relationship between the task-decomposer role and its subordinated team-roles may vary according to features like the degree of agent autonomy or the degree of openness of the MAS.

Figure 5.2 shows an example of a team modelled as a hierarchy of team-roles that is organized as a nested structure of teams, and the straightforward mapping of tasks and capabilities from a task-configuration to team-roles. There is a team-role for each task in the task-configuration, and there is one team for each task-decomposer bound to a task. Each team consist of a "coordinator" team-role responsible for the task being decomposed, plus a set of "subordinated" team-roles, one per subtask. In Figure 5.2, the Team-role 1 (TR1) is assigned to the information-search task, and has to to apply the meta-search task-decomposer capability, which introduces four subtasks. Therefore, TR1 has to coordinate the activity of their subordinated team-roles, one for each subtasks: elaborate-query (TR2), customize-query (TR3), retrieve (TR4) and aggregate (TR5). Moreover, since TR5 is itself assigned to task-decomposer (aggregation), it has to play the coordinator role to interact with their own subordinated team-roles, those associated to their two subtasks, namely elaborate-items (TR6) and aggregate-items (TR7).

5.3.1 Team-roles and team-components

A *team-role* describes the functional, operational and pragmatic aspects required of an agent to occupy a position within a team. The basic information included

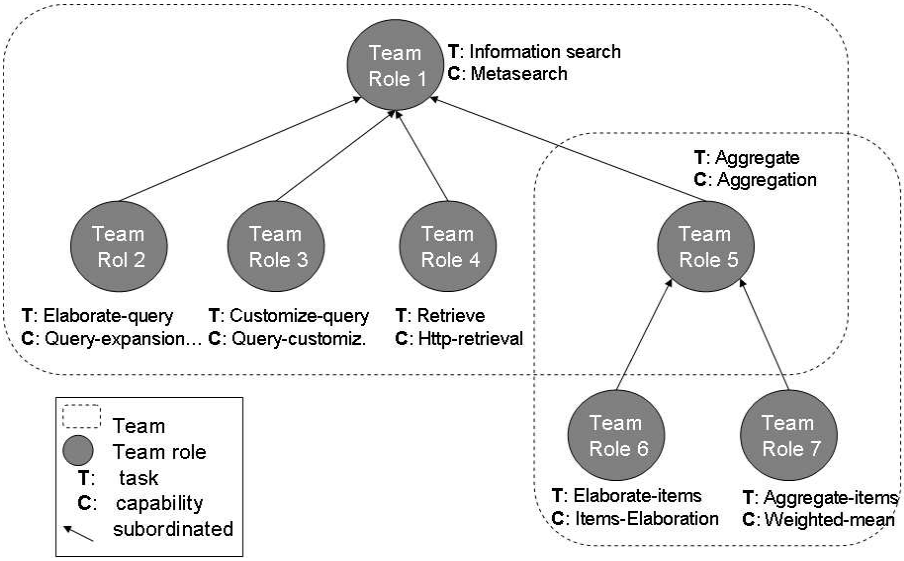


Figure 5.2: Model of a team as a hierarchical team-roles structure

within a team-role comprehends a task to be solved (a goal), a capability to be applied (suitable for the goal task), and interaction elements (communication language and interaction protocols) required to communicate with the agent playing that team-role. In addition, if the capability is a task-decomposer, then the team-role can include information about team members selected for solving each subtask, the communication elements required to delegate each subtask to the selected agent, and optionally a group of agents to keep in reserve.

Formally, a team-role is defined as follows:

Definition 5.1 ***(Team-Role)*** *A team-role is defined as a tuple*

$$\pi = \langle R, I, T, C, M, Com, S, A_S, A_R \rangle$$

where

- R *is a unique team-role identifier,*
- I *is a unique team identifier,*
- T *is a task,*
- C *is a capability,*
- M *is a set of domain-models,*
- Com *is a specification of communication requirements,*
- A_S *is a set of selected agents,*

- A_R *is a set of reserve agents,*
- S *is a subteam, specified as a set of team-components.*

The *team-role identifier* (R) is required to unambiguously identify the position of a team-role in the team hierarchy. The name of the task is not enough to identify a team-role because the task could appear multiple times within a task-configuration. The *team identifier* (I) is required because one agent can participate in multiple teams simultaneously. A team-role includes also the name of the *task* to be achieved (T), the name of the *capability* selected (C) to solve that task (according to a task-capability binding in the task-configuration), and a set of domain-models (M) satisfying the knowledge requirements of the selected capability. In order to instantiate a team model with specific team-members, a team-role includes also two slots to specify a set of agents selected for it (A_S), and a set of reserve agents (A_R). Moreover, a team-role specifies the *communication* (Com) model to be used by both the requester or coordinator of R, and the agent assigned to R.

Finally, a team-role includes a *subteam* feature to be filled only when the team-role's capability (C) is a task-decomposer. A *subteam* is specified as a set of *team-components*, where each team-component holds information about a team-role associated to one subtask. A team-component is defined as follows:

Definition 5.2 ***(Team-Component)*** *A team-component is defined as a tuple*

$$\xi = \langle R, T, A_S, A_R, Com \rangle$$

where

- R *is a unique team-role identifier,*
- T *is a task*
- A_S *is a set of selected agents*
- A_R *is a set of reserve agents*
- Com *is a specification of communication requirements*

A team-component is defined for each subtask introduced by a task-decomposer. The *team-role identifier* (R) determines the precise position of the team-component in the team hierarchy. There is a set of agents selected (A_S) to carry out the team-role, and there is a set of agents to keep in reserve (A_R) for the case that some of the selected agents fail during the Teamwork process. Finally, a team-component includes a specification of the *communication* (Com) required to interact with the agent playing the team-component's team-role (R).

Figure 5.3 shows an example of a team-role assigned to a task-decomposer with two subtasks. Team-role 5 (TR5) is assigned to the Aggregate task, that is bound to the Aggregation capability. This capability is a task-decomposer

that introduces two subtasks: Elaborate-Items and Aggregate-Items. Therefore, the TR5 team-role has a subteam with two team-components, TR6 and TR7, assigned to the Elaborate-Items and the Aggregate-Items tasks respectively.

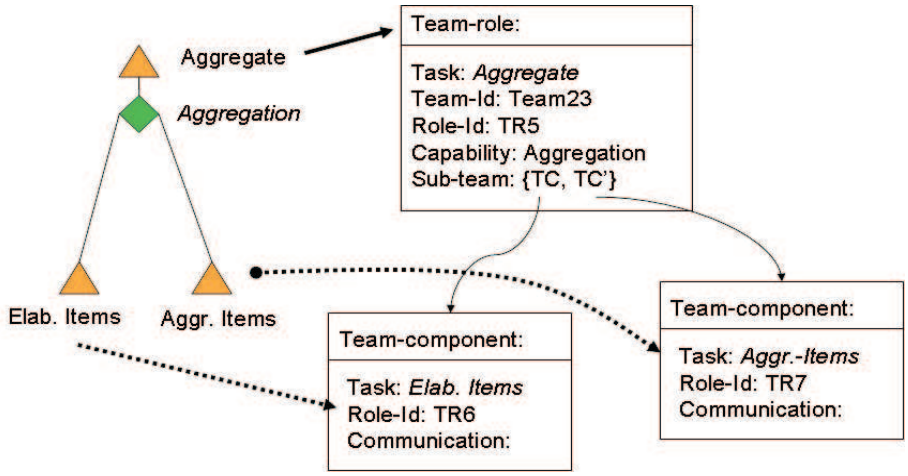


Figure 5.3: From tasks to team-roles and team-components

We note π and Π as a team-role and the set of all the team-roles, and ξ and Ξ as a team-component and the set of all the team-components. We can define now a team as a structure made of interrelated team-roles and team-components, but first we define the *subordinated relation* $\mathbb{S}$ among team-roles as follows:

Definition 5.3 ***(Subordinated)***

$$\mathbb{S}(\pi, \pi') \Leftrightarrow \exists \xi^i \in \pi_S \mid \xi^i_R = \pi'$$

where

- $\pi, \pi' \in \Pi$ *are team-roles;*
- $\pi_S \subseteq \Xi$ *is the subteam of* π *(a set of team-components);*
- $\xi^i \in \Xi$ *is the i-th element of* π_S
- $\xi^i_R \in \Pi$ *is the team-role associated to* ξ^i

Briefly, a team-role is subordinated to another if the first team-role is bound to a team-component contained in the subteam of the second team-role.

Noting $\mathbb{S}^*$ the closure of $\mathbb{S}$ we can now define a *team* as follows:

Definition 5.4 ***(Team)*** *A team is defined for a team-role and a task-configuration*

$$Team(\pi^0, Conf(\kappa)) = \{\pi \in \Pi | \mathbb{S}^*(\pi^0, \pi) \wedge (head(\kappa) = \pi^0_T)\}$$

where

- $\pi^0 \in \Pi$ *is a team leader's team-role, the only team-role that is not subordinated to another;*
- $Conf(\kappa)$ *is a task-configuration,*
- $head(\kappa)$ *is the root task of the task-configuration* $(Conf(\kappa))$*,*
- *and* π^0_T *is the task assigned to the team leader's team-role.*

A *team* is a collection of interrelated team-roles, starting from the team-leader π, that is assigned to the root task of a task-configuration. The ORCAS team model provides an abstract view of the competence required by a group of agents to solve a global problem. Teams are instantiated during the Team Formation process (§5.5) by selecting a set of agents to play each team-role, and a set of agents to keep in reserve.

Team-roles are used during the Team Formation and the Teamwork processes to interchange information among agents. During the Team Formation process, team-roles are used as advertisements or proposals to join a team: team-roles are used to inform potential team members of the tasks to be solved, the capabilities to apply, the knowledge to use, and optionally the terms of commitment and pragmatic issues. Team-roles can also be used by agents to send counter-proposals during the Team Formation process. After finishing the task allocation and the agent selection process, team-roles are used to inform agents about the result of the process.

During the Teamwork process, team-roles are used by team members to know every thing they need to achieve his tasks, to delegate subtasks to other agents, to cooperate with other agents when requested, and even to rescue from unexpected situations like agent failures preempting a task to be achieved by the selected agent, or communication problems avoiding an agent to send or receive messages from other team member. A team-role has the information required by an agent to play a team-role: the name of the task to be solved, the capability to apply, the knowledge to use, and optionally a set of team-components (a subteam) with the information required to delegate the team-role's subtasks to other team members. This information includes the identifiers of subordinated team-roles (those to whom delegate some subtask), a set of selected agents to play each subordinated team-role and a set of agents to keep in reserve for each one, but it does not include information about the capability that should be used by each subordinated team-role.

A team member willing to carry out the task assigned to a team-role during the Teamwork process has to check whether the capability to be applied is a task-decomposer or a skill. On the one hand, if the capability assigned to a team-role is a skill, the agent will engage in communication with the requester. The interaction protocol, the vocabulary and the language to be used will be defined also within a team-role, using the communication feature, as described in §5.4.2. On the other hand, an agent willing to apply a task-decomposer capability has to engage in communication with a requester, just like an agent

applying a skill, but in addition the agent must know which agents are assigned to the task-decomposer's subtasks, so as to request them to carry out their subtasks. The communication to be used so as to delegate a task to another agent is also provided within a team-role structure.

However, an agent applying a task-decomposer does not need to know which capability will be used by the different components of a sub-team, because this information will be sent to those agents separately. In few words, each member of the team knows what to do, which tasks to delegate and to whom, but a team member does not know the precise way a task he delegates to another agent will be solved by it. The last statement is not mandatory, is just a question of information economy, but can be modified to accommodate better to specific situations.

5.4 The ORCAS Agent Capability Description Language

The notion of an *Agent Capability Description Language* (ACDL) has been introduced recently [Sycara et al., 1999a] as a key element in enabling MAS interoperation in open environments. An ACDL is a shared language that allows heterogeneous agents to coordinate effectively across distributed networks. Sometimes, capabilities are referred as "services" and, consequently, an ACDL can alternatively be called an Agent Service Description Language (ASDL).

In the literature, an ACDL is defined as a language to describe both agent advertisements and requests, and is primarily used by middle agents (e.g. brokers and matchmakers) to pair service-requests with service-providing agents that meet the requirements of the request [Sycara et al., 1999b, Sycara et al., 1999a].

Some desirable features for such a language are *expressiveness*, *efficient reasoning* and *easy use*:

- *Expressiveness*: the language should be expressive enough to represent not only data and knowledge, but also the meaning of a capability. Agent capabilities should be described at an abstract rather than implementation level.

- *Efficient reasoning*: inferences on descriptions written in this language should be supported. Automatic reasoning and comparison on the descriptions should be both feasible and efficient.

- *Easy use*: descriptions should not only be easy to read and understand, but also easy to write. The language should support the specification of knowledge requirements (in order to link capabilities to domain knowledge) and the use of ontologies for specifying agent capabilities in a way that favors reuse.

Another important aspect to take into account for designing an ACDL is the idea of enriching capability descriptions with semantic information

[Paolucci et al., 2002]. Semantic markup, which is based on the use of shared ontologies [Guarino, 1997a], improves the matchmaking process and facilitates interoperation.

Although the ORCAS KMF satisfies these requirements, we think an ACDL should bring support to some activities involved in MAS interoperation beyond the discovery of capability providers. An ACDL should facilitate the automation of the following activities, namely discovery, execution, composition and interoperation of capabilities:

Automatic capability discovering (matchmaking): This activity takes the specification of a request and looks for capabilities that are able to satisfy such request. This activity involves the automatic location of capabilities that adhere to requested constraints, which is usually described as a matchmaking process between the request and a library or repository of capabilities (typically hold by a middle agent). An ACDL must allow capability providers to advertise their capabilities to the matchmaker or *yellow pages* service in order to become available for automatic capability discovery. In ORCAS the discovery of capabilities satisfying a problem specification is also achieved through a matchmaking process (§4.2.2), but the ORCAS Knowledge Configuration process goes beyond this requirements and introduces the idea of configuring (designing) a complete MAS-based application (a configured team) that satisfies a specification of stated problem requirements, rather than finding appropriate providers of capabilities suitable for a a single task. The aspects of a capability description required for these activities are the functional descriptions described in the previous chapter: the interface (inputs and outputs) and the competence (preconditions and postconditions), plus the aspects of a capability used to filter out those capabilities which knowledge requirements are not fulfilled.

Automatic capability execution (communication): Having selected a capability, the process of enacting or executing it. Agents should be able to interpret the description of a capability to understand what input is necessary to execute a capability, what information will be returned and which are the effects or postconditions that will hold after applying the capability. In addition, the requester of a capability must know the communication protocol, the communication language and the data format required by the provider of the capability in order to sucessfully communicate with it. Summing up, an ACDL should provide declarative descriptions of both the interfaces and the communication requirements required for executing agent capabilities on request. In ORCAS these aspects of a capability are partially fullfilled by the already described functional aspects of a capability (inputs and outputs, competence, and knowledge requirements), but the communication aspects (interaction protocol and language, and data format) has not been described yet, though we have aforementioned them when talking about the communication feature of a capability (§4.2.1). These aspects are anticipated in the ORCAS KMF, where two features of a capability have been earmarked for further extension of the KM-Ontology into a fullfledged ACDL: the *communication* and the *operational description*. A

proposal for describing these features is introduced later, in §5.4.2 and §5.4.3.

Automatic capability composition (configuration): In order to achieve more complex tasks, capabilities may be combined or aggregated to achieve complex goals that existing capabilities cannot achieve alone. This process may require a combination of matchmaking, capability selection among alternative candidates, and verification of wether the aggregated functionality satisfies the specification of a high-level goal. In ORCAS capabilities are composed during the configuration of a task at the Knowledge Configuration process, using the matching relations introduced in the Knowledge Modelling Framework (§4.2.2), and ensuring that the resulting configuration satisfies the stated problem requirements.

Automatic capability interoperation (coordination): Multiples agents involved in solving a task by applying a composed capability to solve a global task should interoperate between them. Sharing an agent communication language, a common vocabulary and the same interaction protocols are necessary, but not sufficient for cooperation to succeed. In addition to the communication aspects, interoperation among specialized agents during teamwork has to deal with the coordination of agent activities according to the sequencing of tasks and possible task-dependencies. In ORCAS, the information required to coordinate agent behaviors during teamwork is provided by the *operational-description* of the capabilities composing a task-configuration.

The functional description of a capability as provided in the Knowledge-Modelling Framework (Figure 4.8) enables the automated discovery and composition of capabilities (configuration). Nonetheless, the execution of capabilities and the interoperation of multiple agents during teamwork are not supported by the functional description of a capability.

In order to deal with these activities, we have included two extra features to characterize a capability: the *communication* and the *operational description*. Since we keep the knowledge-level aspects of a capability separated from the operational aspects, we avoid including them within the description of a capability in the KMF. This decision allows the ORCAS KMF to be used across different implementations of a MAS and even different types of computational system, like semantic Web services.

The usual approach to overcome the interoperability problems arising in open MAS is to assume a common language and interaction protocols, while the operational aspects of a capability are assumed to be part of the own agent control and thus, they are not declared by an agent when registering its capabilities to a middle agent. While the former elements must be shared by a group of agents in order to work together, open agent environments are encouraged to support more flexible approaches. Therefore, we think the use of declarative descriptions of the communicative and the operational aspects of a capability will support a more flexible architecture where cooperating agents can choose from their repertoire of languages and protocols those that are more appropriate at the moment.

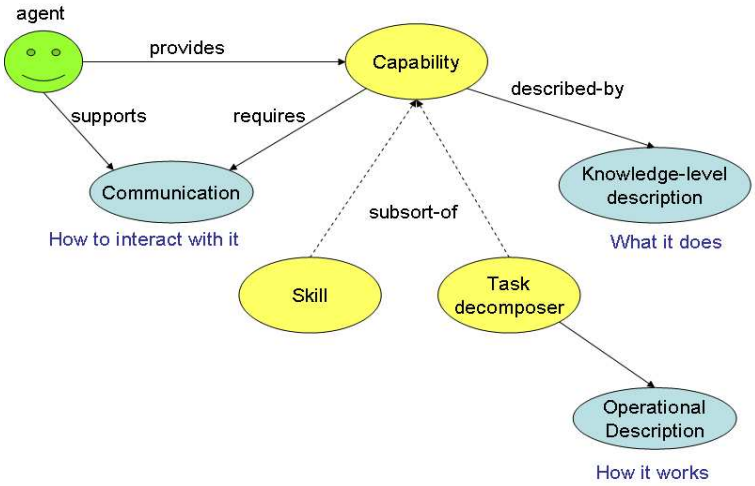


Figure 5.4: Main elements of the ORCAS ACDL concerning capabilities

Figure 5.4 shows the main elements of the ORCAS ACDL. A capability is provided by an agent, and can be either a skill or a task-decomposer. The knowledge-level description of a capability as provided in the Knowledge Modelling Framework (§4.2.1) answers the question "what a capability does?"; that is to say, the KMF provides a functional view of agent capabilities. When a capability is task-decomposer, the ORCAS ACDL provides an operational-description specifying how a task is decomposed into subtasks. In addition, the ORCAS ACDL introduces a communication model that specifies how to interact with an agent so as to request him to apply a capability. The communication model describes the language and the interaction protocol supported by an agent in providing his services to other agents. In general, the same communication model can be used for different capabilities, but some capabilities may not suit some communication models. Therefore, communication models and capabilities are described independently, so as to maximize reuse of both communication models and capabilities. Nonetheless, agents keep the link between the capabilities they provide and the communication models they support over each capability.

The functional aspects of capability are covered by the KMF, which is focused only on those aspects required by the Knowledge-Configuration process. Therefore, those aspects involving agent concepts, like the communication and the operational description, are subjects of the Operational Framework. Specifically, they are presented as two subsections (§5.4.2 and §5.4.3) of the ORCAS ACDL section. concerning the agent approach (e.g. the communication) have been left undefined at the KMF and are addressed at the Operational Framework. But prior to describe these elements in detail we are going to overview the formalism used to specify them: *electronic institutions* [Esteva et al., 2001, Esteva et al., 2002b].

5.4.1 Electronic Institutions

We have stated previously (Chapter 1) our decision of adopting a social, macro-view of Multi-Agent Systems. In particular, we adopt the formal approach of electronic institutions [Esteva et al., 2001, Esteva et al., 2002b] to specifying open agent societies, which is based on a computational metaphor of human institutions.

Human institutions are places where people meet to achieve some goals following specific procedures, e.g. auction houses, parliaments, stock exchange markets, etc. Intuitively, the notion of agent-mediated institution or electronic institution proposes a sort of virtual places where agents interact according to explicit conventions. The institution is the responsible for defining the rules of the game, to enforce them and impose the penalties in case of violation.

An electronic institution, or e-Institution, is a "virtual place" designed to support and facilitate certain goals to be achieved by human and software agents concurring to that place. Since these goals are achieved by means of the interaction of agents, an e-institution must provide the social mediation layer required to achieve a successful interaction: interaction protocols, shared ontologies, communication languages and social behavior rules. An example of such an institution is an Auction House. An Auction House has *institutional agents*, those agents (like the auctioneer) that manage the tasks required for the institution to exist; but Auction Houses are open: they allow other agents (buyers and sellers) to "enter" that place in order to achieve their own goals.

The main goal of the e-Institutions approach is the specification and automatic generation of infrastructures for open agent organizations. Electronic institutions are architecturally-neutral with respect to agents, focused on the macro-level (social view) of agents, and not in their micro-level (internal view).

We are interested on using the concepts proposed by the e-Institutions approach as a way to specify the interaction and coordination needs of teamwork without imposing neither a specific agent architecture, nor a mentalistic theory of cooperation . In electronic institutions, all agent interactions can be reduced to illocutions. Therefore, accountability is expressible in terms of how illocutions are constrained, or what characteristics can be predicated and tested on illocution utterance, and on illocution reception.

A more precise definition of electronic institutions follows [IIIA, 2003]:

> An electronic institution is the computational realization of a set of explicit, possibly enforceable restrictions imposed on a collection of dialogical agent types that concur in space and time to perform a finite repertoire of satisfiable actions.

This definition assumes that agents are "dialogical entities" that interact with other agents within a multi-agent context which is relatively static in ontological terms. We can assume also that agents exhibit rational behavior by engaging in dialogical exchanges, i.e. that agent interactions are systematically linked to illocutions that are comprehensible to participants and refer to a basic shared ontology, and that the exchanges can be (externally) construed as

rational. Moreover, the institution is the real depositary of the ontology and interaction conventions used by the participating agents.

Dialogical agents are entities that are capable of expressing illocutions and react to illocutions addressed to them and, furthermore, only illocutions (and the contextual effects of their associated actions, e.g. commitments to sell a good) constitute observable agent behavior. Individual agents may have other capabilities —perception, intentions, beliefs, etc.—, but we assume that as long as agents interact within the institution, only illocutions are perceivable by other agents (and the meaning and conditions of satisfaction of the associated actions can be objectively established and accounted for within the shared context). Moreover, agents within an institution can only utter illocutions that are consistent with the "role" they are playing. Definitively, the e-Institutions approach is social or exodeitic [Singh, 1998], focused on the macro-level view of Multi-Agent Systems and not on any particular agent architecture.

An e-institution is modelled with the following components [Noriega, 1997, Rodríguez-Aguilar, 1997, Esteva et al., 2001]:

1. *Agent roles:* Agents are the players in an electronic institution, interacting by the exchange of speech acts, whereas roles are standardized patterns of behavior required by agents playing part in given functional relationships. Any agent within an electronic institution is required to adopt some role.

2. *Dialogic framework:* A dialogic framework determines the valid illocutions that can be exchanged among the agents participating in an electronic institution. In dialogical institutions, agents interact through speech acts, thus the institution establishes the acceptable speech acts by defining the ontological elements (the vocabulary) and the agent communication language (ACL). By sharing a dialogic framework, an electronic institution enables heterogeneous agents to meaningfully interact with others.

3. *Communication scenes:* Interaction protocols are articulated through agent group meetings called scenes. A scene defines an interaction protocol among a set of agent roles using a specific dialogic framework. A scene is a formal specification of a pattern of structured communication which constrains the possible patterns of dialogues that can be used by the participating agents adopting one of the roles in the scene (agents have to adopt some role in order to participate).

4. *Performative structure:* A performative structure is a network of connected scenes that captures the relationships among scenes. The specification of a performative structure contains a description of how the different agent roles can move from one scene to another. Furthermore, agents may participate in different scenes, playing different roles at the same time, or engage in multiple instances of the same scene simultaneously.

5. *Normative rules:* Agent actions may have consequences that either limit or enlarge its subsequent acting possibilities. Such consequences are specified

through normative rules, which impose obligations to the agents and affect their possible paths across the performative structure.

ORCAS approaches open agent organizations as virtual, agent based institutions composed of heterogeneous agents playing different roles and interacting by means of speech acts. However, while electronic institutions have been proposed as a way to describe static or predefined organizations, we are rather interested on a more dynamic approach in which the institution is built on-the-fly by putting existing pieces together. While tasks and capabilities where combined during the Knowledge Configuration process to compose a task-configuration, scenes and performative structures are combined and integrated during the Team Formation process. The result is an electronic institution ad-hoc, which provides the communication and the coordination elements required for a team to achieve a global problem according to stated problem requirements.

The following subsections describe the elements required to communicate and coordinate with other agents during the Teamwork process. We will introduce the required notions from electronic institutions and then the way we use those concepts in ORCAS.

5.4.2 Communication

The *communication* aspects of a capability describe the elements required to interact with an agent providing that capability. Interaction is required, basically, to send input data to an agent willing to execute a capability, and to receive back the output produced by the application of a capability from another agent. ORCAS agents are dialogical entities that communicate using speech acts or illocutions; more specifically, we use elements of the electronic-institutions approach to describe the communication aspects associated to a capability. The communication requirements of a capability are specified as scenes using some dialogic framework. A dialogic framework contains the elements for the construction of the communication language, expressions used within the capability communication scenes. Scenes are dialogical patterns of interactions based on the illocutions and vocabulary defined by the dialogic framework. In ORCAS scenes are used to describe the interaction protocols supported by an agent providing some capability. Therefore, scenes are bound to some capabilities within the context of an agent equipped with that capability. The idea here is that of individual agents equipped with a set of capabilities and a set of scenes supported by each capability.

Communication
scenes → set-of *Scene*
dialogic-frameworks → set-of *Dialogic-Framework*

Figure 5.5: The Communication and Communication-scenes sorts

Figure 5.5 shows the Communication sort, used to describe the *communication* features of a capability: a set of *scenes* describing different interaction protocols supported by an agent, and a set of dialogic-frameworks describing the vocabulary and the languages supported by an agent.

The following subsections describe the two elements of the electronic institutions formalism used in ORCAS to describe the communication aspects of a capability: dialogic frameworks and scenes.

Dialogic framework

In open environments agents can be endowed with its own inner language and ontology. In order to allow agents to successfully interact with other agents their languages and ontologies must be put in relation. For this purpose, the electronic institutions approach establishes that agents must share a *dialogic framework* that contains the elements for the construction of the communication language expressions that are required within the institution or within a specific scene. By sharing a dialogic framework, heterogeneous agents can exchange knowledge by means of illocutionary acts.

The electronic institutions formalism defines a dialogic framework as follows [Esteva, 1997]:

Definition 5.5 ***(Dialogic Framework)*** *A dialogic framework is defined as a tuple* $DF = \langle O, L, I, R_I, R_E, R_S \rangle$*, where*

- O *stands for an ontology (vocabulary);*
- L *stands for a content language to express the information exchanged between agents;*
- I *is the a of illocutionary particles;*
- R_I *is a set of internal roles;*
- R_E *is a set of external roles;*
- R_S *is a set of relationships over roles.*

The dialogic framework determines the valid illocutions (I) that can be exchanged between the participants. In order to do so, an ontology (O) that fixes what are the possible values for the concepts in a given domain is defined, e.g goods, participants, locations, etc. Moreover, the dialogic framework defines which are the roles that participating agents may play within the institution or within a particular communication scene the dialogic framework is bound to.

Each role defines a pattern of behavior within the institution. Roles allow to abstract from the individuals agents participating in the electronic institution. This feature is specially important in open environments in which the identity of the agents that could participate in the institution is not known in advance. Furthermore, in open environments agents may change over time, since new

agents may join the institution and agents already in the institution may come to leave. For this reason, all the actions that can be done within an institution are associated to roles, and not to individual agents. Intuitively we can think of roles as agent types characterized by a set of actions allowed for that type. For instance, within an auction, an agent playing the buyer role is capable of submitting bids, while the agent playing the auctioneer role can offer goods at auction. In order to take part in an electronic institution, an agent is obliged to adopt some role(s). An agent playing a given role must conform to the pattern of behavior attached to that particular role. However, all agents adopting a specific role are guaranteed to have the same rights, duties and opportunities.

A dialogic framework distinguishes internal roles (R_I) from external roles (R_E). Internal roles define the roles to be played by staff agents, which are equivalent the employees of a human institution. Those agents are in charge of guaranteeing the correct functioning of an institution. For instance, an auctioneer is in charge of auctioning goods following the specified protocol and the buyer admitter is in charge of guaranteeing that only buyers satisfying the admission conditions are allowed to participate.

Two types of agent relationships over roles can be specified, namely: *superclass* and *static separation of duties* (SSD). *Superclass* relationships indicate whether a role belongs to a more general class. If a role r is a superclass of another role r' ($r \succeq r'$), then an agent playing r is enabled to play r'. However, since agents can play several roles at the same time, role relationships standing for conflict of interests must be defined with the purpose of protecting the institution against an agent's malicious behavior. For instance, in an auction house the auctioneer and the buyer roles are mutually exclusive. A static separation of duties policy is defined to avoid two mutually exclusive roles of being authorized to the same agent. The static separation of duties is defined as the relation $ssd \subseteq Roles \times Roles$. A pair $(r, r') \in ssd$ denotes that r, r' cannot be authorized to the same agent. See [Esteva et al., 2001] for an enumeration of the requirements for the ssd relation and some inferred properties.

The content language (L) allows for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. The propositions built with the aid of the content language are embedded into an "outer language", the communication language(CL), which expresses the intentions of the utterance by means of the illocutionary particles.

Expressions in the communication language are constructed as formulae of the type $(\iota\,(\alpha_i\,\pi_i)\,(\beta)\,\varphi\,\tau)$ where ι is an *illocutionary particle*, α_i is a term which can be either an agent variable or an agent identifier, π_i is a term which can be either a role variable or a role identifier, β represents the addressee(s) of the message (which can be an agent or a group of agents), φ is an expression in the *content language* and τ is a term which can be either a time variable or a time constant. The CL allows to express that an illocution is addressed to an agent, to all the agents playing a role or to all the agents in the scene.

Notice that a dialogic framework determines the ontological elements and the valid elements for constructing expressions in the communication language.

Thus a dialogic framework must be regarded as a necessary ingredient to specify scenes.

Using dialogic-frameworks in ORCAS: the Teamwork ontology

Given a capability C, one can think that some of the ontological elements required to interact with the agent providing C are those used to specify C, like the signature-elements used to specify the input and the output. Nonetheless, we have preferred to keep the knowledge-level elements away from the operational and communication aspects so as to maximize the reuse of these elements too. In order to do that, we have decided to specify the communication of a capability independently of other features of the capability: we specify the communication aspects using generic, easy to reuse concepts, like the notion of input and output, and not in terms of a particular type of input (a signature-element).

Since the ORCAS framework aims to maximize reuse (of both capabilities and communication elements), thus we try to impose as few requirements as possible to agents willing to cooperate, thus we have imposed a minimum set of concepts to be understood by agents in order to participate in an ORCAS e-Institution. These concepts should be shared by all the members of a team in order to cooperate, thus they are explicitly represented as an ontology. This ontology consist of the previously presented notions of Team-role (§5.1) and Team-component (§5.2), plus some concepts relating the different types of messages that can be exchanged during the Teamwork process. A basic model of communication for teamwork includes the following type of messages:

- *Perform*: requests to solve the tasks associated to a specific team-role.
- *Result*: messages containing the results of having performed some task.
- *Done*: messages to confirm that some request has been achieved
- *Refusal*: messages to inform that the requested petition won't be carried on.
- *Failure*: messages to inform of some failure occurred while performing the tasks associated to a team-role.

Figure 5.6 shows the basic concepts included in the Teamwork ontology, and the features characterizing each concept. These concepts and their features are defined as sorts in the Teamwork ontology, and are used within scenes to constrain the type of messages allowed in the following way: only messages complying with the illocutionary schemas defined by the scene are permitted. For example, if an illocutionary schema specifies that a message should have a content of the type *Perform*, then only messages with that type of content would be allowed. The sort Any subsumes any other sort and is used to allow further specialization by developers, for instance, the reason for a refusal or the error code when informing about a failure.

Teamwork ontology	
Team-role	Definition 5.1
Team-component	Definition 5.2
Perform	team-id:Symbol, team-role:Symbol, input-data:Signature-element
Result	team-id:Symbol, team-role:Symbol, output-data:Signature-element
Done	team-id:Symbol, team-role:Symbol
Refusal	team-id:Symbol, team-role:Symbol, reason:Any
Failure	team-id:Symbol, team-role:Symbol, error:Any

Figure 5.6: Basic Teamwork concepts

Capabilities should be specified independently of other capabilities in order to maximize their reuse and facilitate their specification by third party agent developers. In the general case, agent developers do not know a priori the tasks that could be achieved by an agent capability, since teams are formed on-demand according to specified problem requirements, and thus the same capability could be used to solve different tasks (as far as the the capability is suitable for the task, as defined by a task-capability matching relation) or the same task in the context of a different task-decomposition. As a consequence, the team roles an agent can play using a capability are not known in advance. Therefore, the roles used to specify the communication scenes of a capability cannot be specified in terms of specific team-roles.

Our approach to overcome the former difficulty is to specify the communication aspects of a capability in terms of abstract, generic roles, rather than specific team-roles. As already explained, ORCAS teams are hierarchically organized according to task-configurations, and thus the teamwork itself can be easily coordinated using a hierarchical communication style. There are agents decomposing a task into subtasks and requesting other agents to solve some of the subtasks. An agent that applies a task-decomposer capability to solve a task is responsible for delegating subtasks to other agents, receiving the results, and performing intermediate data processing between subtasks. In such a scenario, we can establish an abstract communication model with two basic roles:

1. the *coordinator* role is adopted by an agent willing to decompose a task into subtasks, requesting other agents to carry on the different subtasks in an appropriate order, receiving the different results, and obtaining the final result of the task; and

2. the *operator* role, one the other side, is adopted by the agent having to perform a task on demand, using the data provided by another agent that acts as coordinator, and having to bring the result back to the coordinator.

The operator and the coordinator roles are defined as keeping a static separation of duties (SSD) relationship, so as to avoid an agent to adopt both roles simultaneously within the same scene. However, the same agent can act as coordinator and operator simultaneously, but playing those roles in different scenes.

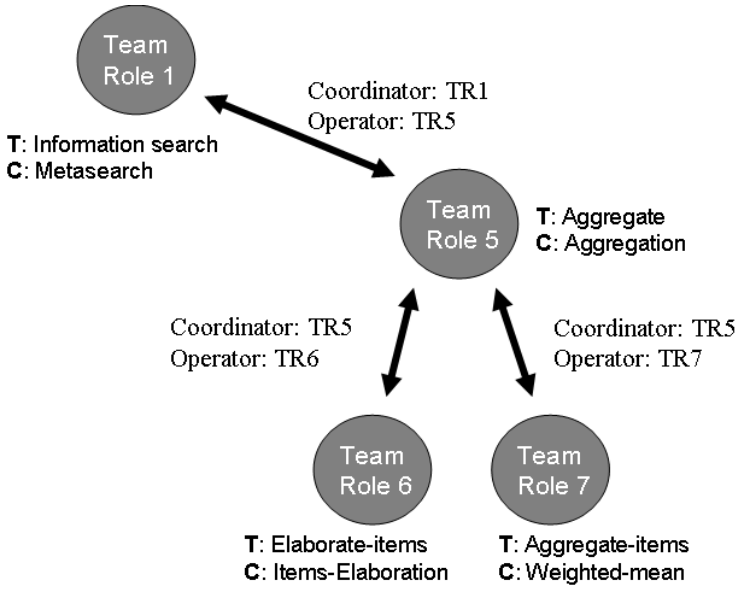


Figure 5.7: Example of team-role relations and role-policy for Teamwork

This situation occurs when a team-role is neither the root neither a leave in the team-roles hierarchy. The agent playing such a team-role has to adopt the operator role to communicate with the agent assigned the task-decomposer on top of it (one level above in the team hierarchy). Nonetheless, in order to finish its task, this agent has to communicate with other agents assigned to its own subtasks, and adopting the coordinator role itself. Figure 5.7 shows an example of such a situation. The agent playing Team-role 5 (TR5) has to act as operator to communicate with TR1. Notice that TR5 (the agent playing TR5) has to solve the task Aggregate using the input data received from TR1, and send the result back to TR1; but in order to do that, TR5 must delegate its own subtasks (elaborate-items and aggregate-items) to its subordinated team-roles, TR6 and TR7. In order to do that, TR5 has to communicate with TR6 and TR7 acting himself as the coordinator, and TR6 and TR7 as the operator (each one in a different scene).

Moreover, we introduce another role that is a superclass of both the coordinator and the operator roles, the *Problem-Solving Agent* (PSA) role.

$$(PSA \succeq Coordinator) \wedge (PSA \succeq coordinator) \tag{5.1}$$

Consequently, any agent enabled playing a PSA role can play also the coordinator and the operator roles (albeit never in the same scene instance). Figure 5.8 depicts the relationships among the basic ORCAS roles. The PSA role is a superclass of both the coordinator and the operator roles, and there is a static

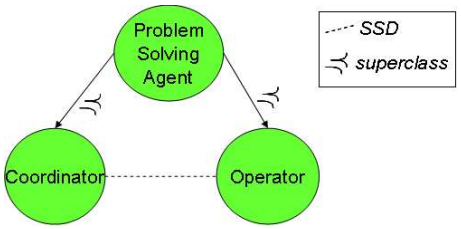


Figure 5.8: Basic roles and role relationships

separation of duties between the coordinator and the operator roles.

Figure 5.9 summarizes the specification of the dialogic framework in the ORCAS ACDL: A shared Teamwork ontology, and the coordinator and operators as the only roles. There is a static separation of duties (SSD) between the coordinator and the operator roles. This relation means that an agent cannot be coordinator and operator within the same scene, since it has no sense for an agent to communicate with himself. There are no internal roles and both the content language and the illocutionary particles remain open (the example shows the typical illocutions and specifies NOOS as the content-language).

ontology	***Teamwork-ontology***
illocutionary-particles	*e.g. (request inform agree refuse inform failure)*
internal-roles	$\emptyset$
external-roles	*(**coordinator operator**)*
social-structure	*(**coordinator SSD operator**)*
content-language	e.g.*NOOS*

Figure 5.9: Dialogic frameworks in the ORCAS ACDL

Scenes

A scene is the main element used to describe the communication features of a capability, it describes what is commonly known as an agent interaction protocol. The same capability can support different interaction protocols, and thus a scene is required to specify each interaction protocol.

Recall that a scene is a conversation protocol shared by a group of agents playing some roles. More precisely, a scene defines a generic pattern of conversation among roles. Any agent participating in a scene has to play one of its roles. A scene is generic in the sense that it can be repeatedly played by different groups of agents, in the same sense that the same theater scene can be performed by different actors playing the same roles.

Electronic institutions use finite state machines (FSM) to specify scenes[1],

[1]An account of the reasons to adopt such a formalism is found in [Esteva, 1997], while another examples on using FSMs to specify agent conversations can be found in

which are represented by finite, directed graphs. A scene is defined as follows in the electronic institutions formalism [Esteva, 1997]:

Definition 5.6 ***(Scene)*** *A scene is a tuple:*

$$s = \langle R, DF, W, w_0, W_f, (WA_r)_{r \in R}, (WE_r)_{r \in R}, \Theta, \lambda, min, Max \rangle$$

where

- R *is the set of roles of the scene;*
- DF *is a dialogic framework (Definition 5.5);*
- W *is a finite, non-empty set of scene states;*
- $w_0 \in W$ *is the initial state;*
- $W_f \subseteq W$ *is the non-empty set of final states;*
- $(WA_r)_{r \in R} \subseteq W$ *is a family of non-empty sets such that* WA_r *stands for the set of access states for the role* $r \in R$*;*
- $(WE_r)_{r \in R} \subseteq W$ *is a family of non-empty sets such that* WE_r *stands for the set of exit states for the role* $r \in R$*;*
- $\Theta \subseteq W \times W$ *is a set of directed edges;*
- $\lambda : \Theta \longrightarrow L$ *is a labelling function, where* L *can be a timeout, an illocution scheme or an illocutions scheme and a list of constraints;*
- min, Max *are two functions that return respectively the minimum and maximum number of agents that can play a role* $r \in R$*;*

The nodes of the scene graph represent the different *states* (W) of the conversation, and the directed *edges* (Θ) connecting the nodes are labelled (λ) with the actions that make the scene state evolve: illocution schemes and timeouts. The graph has a single *initial state* (w_0, non-reachable once left) and a set of *final states* (W_f) representing the different endings of the conversation (there is no edge connecting a final state to another state).

A scene allows agents either to join it or leave it at specific states during an ongoing conversation, depending on their role. For this purpose, the sets of *access* (WA_r) states and *exit* states (WE_r) are differentiated for each role.

Normally the correct evolution of a conversation protocol requires a certain number of agents for each role involved in the scene. Thus, a *minimum* (min) and a *maximum* (Max) number of agents per role is defined and the number of agents playing each role has to be always between them.

The final states have to be an exit state for each role, in order to allow all the agents to leave when the scene is finished. On the other hand, the initial

[Barbuceanu and Fox, 1995, Nodine and Unruh, 1999, d'Inverno et al., 1998]

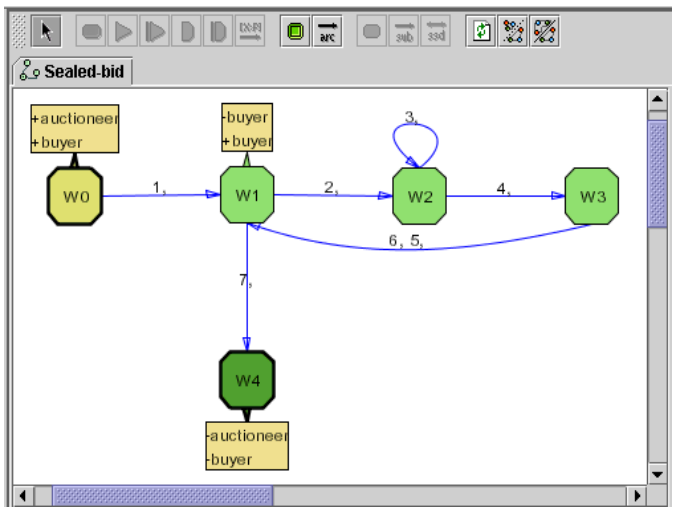
state has to be an access state for the roles whose minimum is greater than zero, in order to start the scene.

The information exchanged between agents is expressed in the form of illocution schemes from the scene dialogic framework. In order for the protocol to be generic some details have to be abstracted. This means that state transitions cannot be labelled by grounded illocutions. Instead, illocutions schemes must be used, where the terms referring to agents and time are variables while the other terms can be variables or constants.

The other element that can label an edge is a timeout. Timeouts trigger on transitions after a given number of time units have passed since the state was reached. This is specially important for robustness —to evolve from states where agents dying and hence not talking any more, or where agents trying to foot-drag the other agents by remaining silent, could block the scene execution.

In addition to defining the valid sequences of illocutions that agents can exchange, a scene establishes the conversation context. Context is a fundamental aspect that humans use in order to interpret the information they receive. The same message in different contexts may certainly have a different meaning. Thus, a scene establishes what can be said, by who, and to whom, and allows to specify how past interactions may affect the future evolution of the conversation. The contextual information may restrict the valid messages in a certain state of the conversation. For instance, imagine a scene auctioning goods following the English auction protocol: as bids are submitted by buyers the valid bids for them are reduced to bids greater than the last one. That is to say, each submitted bid reduces the valid illocutions that buyers can utter, although the scene may continue in the same state. Such contextual information is encoded as constrains, which are used to restrict the set of values to create new bindings of the variables in the illocution schemes, as well as the paths that a scene conversation can follow. The reader is referred to [Esteva, 1997] for a detailed account on the use of variables and constrains in the electronic institutions formalism.

A scene has both a textual and a graphical representation. Figure 5.10 shows an example of an auction scene specifying a sealed-bid protocol. In this scene the participating agents can play the auctioneer and buyer roles. The graph depicts the states of the scene, along with the edges representing the legal transitions between scene states which are labelled either with illocution schemes of the communication language (according to elements defined within a dialogic framework) or with timeouts. Notice that apart from the initial and final states, the $w1$ state is labelled as an access and exit state for buyers, which means that buyers can leave and new buyers might be admitted between bidding rounds. Variable identifiers appearing in the illocution schemes can start with either '?' or '!', which is used to differentiate whether the variable can be bound to a new value or must be substituted by its last bound value.

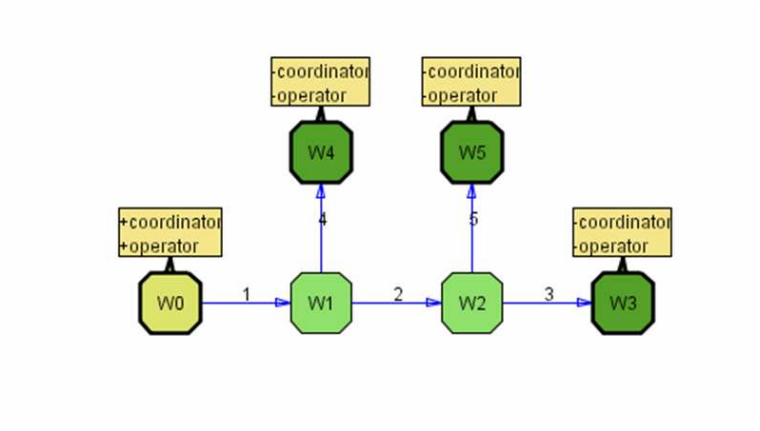


1 $(inform\ (?x\ auctioneer)\ (buyer)\ open\_auction(?r))$
2 $(inform\ (!x\ auctioneer)\ (buyer)\ start\_round(?good\_id, ?bidding\_time, ?reserve\_price))$
3 $(commit\ (?y\ buyer)\ (!x\ auctioneer)\ bid(!good\_id, ?offer))$
4 $!bidding\_time$
5 $(inform\ (!x\ auctioneer)\ (buyer)\ sold(!good\_id, ?price, ?winner))$
6 $(inform\ (!x\ auctioneer)\ (buyer)\ withdrawn(!good\_id))$
7 $(inform\ (!x\ auctioneer)\ (buyer)\ end\_auction(!r))$

Figure 5.10: Specification of a sealed-bid auction protocol

Using scenes in ORCAS

In the electronic institutions formalism, when a new institution is defined, there is a global view of the system and thus it is possible to define in advance all the roles that can be played by the participating agents. Scenes in ORCAS are used to describe the patterns of interaction required to delegate a task to other agent, exclusively from the point of view of the agent providing the capability required to achieve that task. An agent providing a capability does not know in advance the potential team-roles it can get to play using that capability. Consequently, a scene describing the communication requirements of a capability can not be specified in terms of team-roles. The ORCAS approach to deal with issue is to define scenes in terms of two generic roles: the *coordinator* and the *operator* roles. The agent applying a task-decomposer and willing to delegate some subtask takes the coordinator role to communicate with each of the agents assigned to a subtask within the task-decomposer, engaging in a new scene for each task being delegated. Realize that the communication between an agent applying a task-decomposer and an agent responsible of one subtask is necessary only both the agents are the same; otherwise, if the agent assigned a task-decomposer were the same assigned a subtask, then there is no need for communication for that subtask.



1 *(request (?x Coordinator) (?y Operator) perform(?team-role ?input))*
2 *(agree (!y Operator) (!x Coordinator) perform(!team-role !input))*
3 *(inform (!y Operator) (!x Coordinator) result(!team-role ?ouput))*
4 *(refuse (!y Operator) (Coordinator null) perform(!team-role !input))*
5 *(failure (!x Operator) (!y Coordinator) perform(!team-role !input))*

Figure 5.11: Request-Inform protocol described by a scene

Figure 5.13 shows a scene specifying a FIPA-like Request-Inform protocol for a capability C. There are two roles, the *coordinator* and the *operator* roles. The coordinator role is played by the agent that has to delegate a task T to the agent providing C (assuming that T can be solved by C). The operator

role is played by the agent providing C. The coordinator is the initiator of the scene, that begins with the coordinator sending a "request" message to the operator. That message contains the identifier of the team-role to be played by the operator, and the data to be used as input. The operator checks whether it is assigned to that *team-role*, and sends either an "agree" or a "refuse" message accordingly (if an agent is assigned to a team-role, it is supposed to agree, owing to the commitment implicit on an agent accepting a team-role during the Team Formation process). The operator agent holds the information required to carry out its accepted team-roles, as that information was provided during the Team formation process; therefore, the operator can solve the task assigned to that team-role by applying the capability bound to it, using the data provided as input by the coordinator, and the selected domain knowledge. If the capability is applied to the input data successfully, then the operator sends the result to the coordinator with an "inform" message, otherwise the operator sends a "failure" message. There are three final states that are reached when the operator refuses the request from the coordinator ($w4$), the application of the capability fails ($w5$), or it ends successfully ($w3$).

A wide range of communication styles can be specified using scenes, from very simple protocols involving two roles to very complex interaction protocols with several agent roles, time-outs, transition constrains and so on, thus giving quite expressiveness to developers. Nevertheless, using a reduced set of standardized basic protocols is encouraged in ORCAS to maximize capability reuse. In order to have an intuitive idea of possible styles of interaction we can consider the basic interaction protocols defined for Web services, which are called operations in WSDL and processes in DAML-S [The DAML-S Consortium, 2001]. There are four basic types of "operations" according to these proposals:

- *request-response* operation (an atomic process with both inputs and outputs in DAML-S);
- *one-way* operation (an atomic process with inputs but no outputs);
- *notification* operation (an atomic process with outputs, but no inputs);
- *solicit-response* operation (a composite process with both outputs and inputs, and with the sending of outputs specified as coming before the reception of inputs).

The *request-response* operation corresponds to the request-inform protocol showed in Figure 5.13 as an example of a scene. The *one-way* and the *notification* operations can be specified by the same scene but with different edge labels, as showed in Figure 5.12. Finally, the solicit-response operation requires some minor changes in the scene; there is required a new state, w_6, and a new transition (Transition 6) from $w5$ with an inform message to be send by the Coordinator to the Operator, containing the input.

Summing-up, scenes provide a flexible and precise (not ambiguous) way of defining interaction protocols to communicate with agents in order to delegate

	One-way operation (only input)
1	*(request (?x Coordinator) (?y Operator) perform(?team-role ?input))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role !input))*
3	*(inform (!y Operator) (!x Coordinator) done(perform(!team-role !input))*
4	*(refuse (!y Operator) (Coordinator null) refusal(perform(!team-role !input) reason)*
5	*(failure (!x Operator) (!y Coordinator) failure(perform(!team-role !input) reason)*

	Notification: (only output)
1	*(request (?x Coordinator) (?y Operator) perform(?team-role))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role))*
3	*(inform (!y Operator) (!x Coordinator) result(!team-role !output))*
4	*(refuse (!y Operator) (!xCoordinator) perform(!team-role))*
5	*(failure (!x Operator) (!y Coordinator) perform(!team-role))*

	Solicit-Response (both input and output, but input cames after the output is provided
1	*(request (?x Coordinator) (?y Operator) perform(?team-role))*
2	*(agree (!y Operator) (!x Coordinator) perform(!team-role))*
3	*(inform (!y Operator) (!x Coordinator) result(!team-role !output))*
4	*(inform (!x Coordinator) (!x Operator) response(!team-role !input))*
5	*(refuse (!y Operator) (Coordinator null) inform(!team-role ?input))*
6	*(failure (!x Operator) (!y Coordinator) perform(!team-role))*

Figure 5.12: Variations and alternatives to the Request-Inform protocol

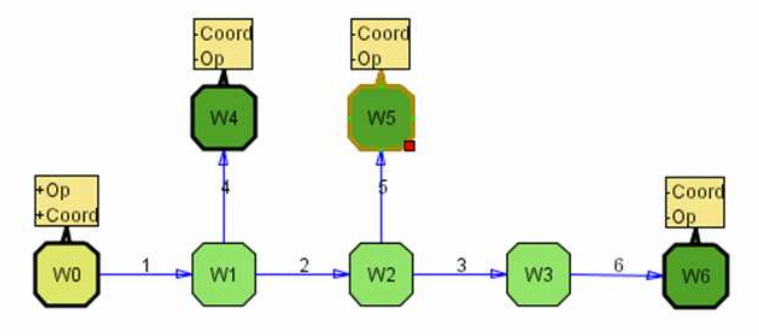


Figure 5.13: Solicit-Response protocol described by a scene

tasks to other agents. Every time an agent has to delegate a task to another in the context of a teamwork process, it has to engage in communication with the agent assigned to the task's team-role. This communication will follow the specification of a scene supported by the agent providing that capability. The decision on which specific scene is to be used between two agents is negotiated during the Team Formation process, as explained in §5.5.

5.4.3 Operational description

The purpose of the *operational description* is to specify the data and control flow among the subtasks of a task-decomposer. Control flow languages are based on the notion of control constructs like iteration, parallelism, branching conditions and so on. Control flow modelling has been a research area that attracted significant interest in the last decade. Nevertheless, little consensus has been reached as to what the essential ingredients of a control flow specification language should be, and there are notable differences in the expressive power of control flow specification languages [Kiepuszewski, 2002].

Since our approach is based on Knowledge Modelling frameworks we have considered some proposals from the Knowledge Modelling community, like KARL [Fensel et al., 1998a] and Modal Change Logic [Fensel et al., 1998b], but we found that these proposals rely on a sequentiality assumption that is not appropriate for MAS. Therefore, it seems more appropriate to use agent concepts for describing the interaction among subtasks in order to deal with parallelism. Such a language must capture dependency relationships, temporal relationships, and parallelism in order to support the team configuration during the Team Formation process, and the coordination of team mates during the Teamwork process. In order to describe these aspects, we will continue using the concepts on electronic institutions: specifically, we will use the notion of a performative structure to describe the operational description of task-decomposers.

We want to increase the reusability of capabilities by describing the operational description of task-decomposers from a compositional approach, maximizing the reuse of capabilities by keeping them separated from both the tasks and the domain models. In addition, we want to use a formalism supporting parallelism and allowing for the specification of synchronization points, and multiple task instantiation (i.e. tasks that can be played several times during the same teamwork process).

In order to deal with these issues, the ORCAS ACDL proposes the specification of the operational description as a task network, using the concept of a *performative structure* from the electronic institutions formalism. The point is to describe the operational description of a task-decomposer as a composition of several scenes, where each scene corresponds to a scene describing the communication between the agents playing the coordinator and the operator roles for that capability during the Teamwork process.

While a scene models a particular multi-agent dialogical activity, more complex activities can be specified by establishing relationships among scenes. This issue arises when conversations are embedded in a broader context, such as, for

instance, organizations and institutions. If this is the case, it does make sense to capture the relationships among scenes. For these purpose a performative structure defines which are the scenes of the electronic institution and the *role flow policy* among them. That is to say, *how* the agents can move among the different scenes depending on their role, and *when* new scenes have to be started, taking into account the relationships among the different scenes.

Performative structure

A performative structure is a network of connected scenes that captures the relationships among scenes. The specification of a performative structure contains a description of how the different agent roles can move from one scene to another. More formally, a performative structure is defined as follows [Esteva, 1997]:

Definition 5.7 ***(Performative Structure)*** *A performative structure is a tuple*

$$PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, C, \mu \rangle$$

where

- S *is a finite, non-empty set of typed scenes, where each scene is defined by a name* (S_{name}) *and a type* (S_{type}) *(Def. 5.6);*
- T *is a finite and non-empty set of transitions;*
- $s_0 \in S$ *is the* initial *scene;*
- $s_\Omega \in S$ *is the* final *scene;*
- $E = E^I \bigcup E^O$ *is a set of edge identifiers where* $E^I \subseteq S \times T$ *is a set of edges from scenes to transitions and* $E^O \subseteq T \times S$ *is a set of edges from transitions to scenes;*
- $f_L : E \longrightarrow V$ *maps each edge to an edge label V, represented as a disjunctive normal form (DNF)*[2] *in which literals are pairs composed of an agent variable and a role identifier representing an edge label;*
- $f_T : T \longrightarrow \mathcal{T}$ *maps each transition to its type;*
- $f_E^O : E^O \longrightarrow \mathcal{E}$ *maps each edge to its type;*
- $C : E^I \longrightarrow CONS$ *maps each edge to a expression representing the edge's constraints.*
- $\mu : S \longrightarrow \{0, 1\}$ *establishes whether a scene can be multiply instantiated at execution time;*

[2]Disjunctive Normal Form or DNF is a method of standardizing and normalizing logical formulae. A logical formula is considered to be in DNF if and only if it is a single disjunction of conjunctions. More simply stated, the outermost operators of the formula are all ORs, and there is only one level of nesting allowed, which may only contain literals or conjunctions of literals

A performative structure contains a set of typed *scenes* (S). The scene type is the specification of the scene according to Definition 5.6), thus different scenes within a performative structure can refer to the same type of scene. There are two scenes defined as the initial (s_0) and final (s_Ω) scenes. Relationships among scenes are specified as *transitions* (T) agents must traverse in order to move from one scene to another, and edges going from scenes to transitions (incoming edges, E^I), and from transitions to scenes (outgoing edges, E^O). In order to move from one scene to another, an agent has to progress through a *transition* (direct connections between scenes are forbidden). In general, the activity represented by a performative structure can be depicted as a collection of multiple, concurrent scenes, and agents navigating from scene to scene constrained by transitions. A performative structure can specify also whether a scene can be multiple instantiated or not at execution time (μ).

The edges of a performative structure are labelled so as to specify which agents can progress through an edge depending on their roles. These labels are expressed as conjunctions and disjunctions of pairs composed of an agent variable and a role identifier. The role identifier determines which type of agent is allowed to follow the edge, while agent variables are used to differentiate among agents playing the same role. For instance, an edge labelled with $(x\,R_1) \wedge (y\,R_2)$ means that this edge can be followed only by pairs of agents where one of them is playing the role R_1 and the other is playing the role R_2. On the other hand, an edge labelled with $(x\,R_1) \vee (y\,R_2)$ means that any agent playing either the role R_1 or the role R_2 can progress through that edge. The scope of an agent variable includes all the incoming and outgoing edges of a transition. That is to say, if an agent reaches a transition following an incoming edge labelled with $(x\,R_1)$, it can only leave the transition by following those outgoing edges containing the variable x in their label. However, there is a relation between the agent variables labelling the incoming and the outgoing edges of a scene. A conjunction over a incoming edge (from a scene to a transition) means that the agents have to leave the scene together (and reach the transition together too), whereas a conjunction labelling an outgoing edge (from a transition to a scene) means that the agents must enter the target scene together, that is to say, agents must enter into the same scene instance.

There are two types of transitions (f_T) according to how agents coming from several edges can progress through them:

- **AND** transitions establish synchronization points and parallelism. Agents reaching a transition from several incoming edges have to wait for agents coming from all the incoming edges in order to progress through the transition, and must follow all the outgoing edges where they appear (the variables they are bound to).

- **OR** transitions allow agents to progress through them in an asynchronous way and are used to define choice points. Agents reaching an OR transition can progress through it without waiting for other agents, and are allowed to choose which outgoing edge to follow when leaving the transition.

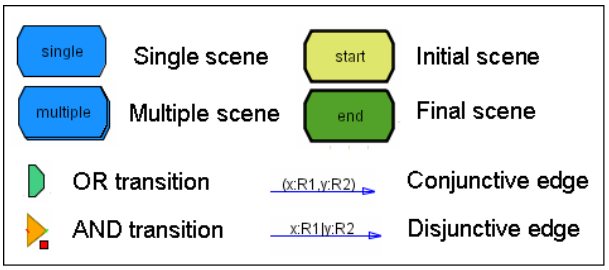


Figure 5.14: Graphical elements used to specify a performative structure

Figure 5.14 shows the graphical elements used to specify a performative structure: scenes, transitions and edges. There are different symbols to distinguish the initial and final scenes from other scenes, and to differentiate whether a scene can be multiple instantiated or not (single). Notice there are two types of transitions, AND and OR; and two ways of labelling an edge, conjuntion and disjuntion.

An example of a performative structure for an agoric market is shown in Figure 5.15 (extracted from [Esteva, 1997]). The root scene is the Admission scene, where any agent enters the institution. Buyers and sellers can move from the Admission scene to the Agora scene, where they can try to buy and sell goods. When a buy/sell operation is agreed, both the involved buyer and seller together meet with an accountant agent in the Settlement scene to formalize the operation. Finally, agents can exit the institution by reaching the Departure scene.

Using performative structures in ORCAS

Our approach to specify the operational description of a task-decomposer is based on performative structures, with some distinctive features.

A first feature characterizing the use of performative structures to specify the operational description of a task-decomposer arises from the fact that the precise team-role applying a task-decomposer is not known, because a task-decomposer is defined for a capability, independently of any particular task that can solved applying that capability. Our approach to overcome this problem is the use of two generic roles, as explained in §5.4.2 and §5.4.2: the coordinator and the operator roles. Therefore, when defining an operational description as a performative structure, we know that the agent providing a task-decomposer adopts the role of the coordinator, while the agents assigned to every subtask adopt the operator role, and thus, the performative structure can be defined in terms of these generic roles.

The second feature distinguishing a performative structure in ORCAS from the electronic institutions approach results from the decoupling of tasks and capabilities. In ORCAS, each scene within the performative structure corresponds

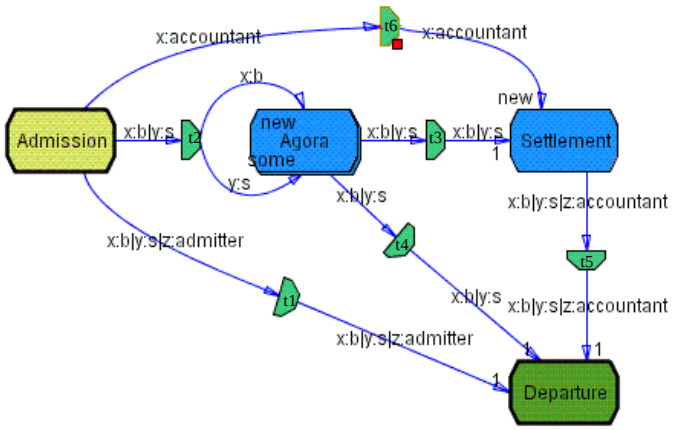


Figure 5.15: Performative structure of an agora

to an interaction protocol describing the communication required between a task-decomposer team-role acting as a coordinator, and a subordinated team-role acting as operator. In other words, each scene within the performative structure refers to a task to be delegated to another agent. In ORCAS, each task (T) is solved using a capability (C); therefore the coordinator agent has to communicate with the agent providing C, using one of the scenes supported by that agent (acting as operator) over C. Consequently, the specific scenes to be used within a performative structure must be decided before starting the Teamwork process. In ORCAS this process is carried over during the Team Formation process. The goal is to select the scenes from those supported from both the provider and the requester of a capability. We note A^C an agent providing C and A^C_S as the set of scenes it supports over C. The provider of a capability must play the operator role and the requester must play the coordinator role. Both the coordinator and the operator must follow the same scene in order to communicate, and as a consequence, the scene must be chosen out of the scenes supported by both agents (the intersection of two set of scenes, the scenes supported by the coordinator, and the scenes supported by the operator).

In order to specify the operational description of a task-decomposer, we adapt the notion of a performative structure to fit better in the ORCAS framework. Specifically, the constraints and the edge typing functions are not used in an operational description, the scenes are not typed ($\dashv$), and two scenes called Start and End are defined as the initial and final scenes. More formally, an operational description in ORCAS is defined as follows [Esteva, 1997]:

Definition 5.8 ***(Operational description)*** *A operational description is defined for a task-decomposer D as a tuple*

$$OD(D) = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, \mu \rangle$$

where

- *S is a set of untyped scenes named after the subtasks introduced by the task-decomposer, plus an initial and a final scene called* Start *and* End *respectively;*
- *$s_0 \in S =$ Start (the initial scene);*
- *$s_\Omega =$ End (the final scene);*
- *T is a finite and non-empty set of transitions;*
- *$E = E^I \bigcup E^O$ is a set of edge identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes;*
- *$f_L : E \longrightarrow V$ maps each edge to an edge label V, represented as a disjunctive normal form (DNF) over pairs composed of an agent variable and a role identifier representing an edge label;*
- *$f_T : T \longrightarrow \{AND, OR\}$ maps each transition to its type;*
- *$\mu : S \longrightarrow \{0, 1\}$ establishes whether a scene can be multiply instantiated at execution time;*

Notice that the set of scenes of an operational description has no type, that is to say, scenes are not bound to a scene specification, but they have just a name. Moreover, scenes are named as the subtasks of the task-decomposer, plus two scenes called Start and End.

Figure 5.16 shows an example of a performative structure specifying the operational description of the Aggregation task-decomposer, which decomposes a task into two subtasks: Elaborate-items and Aggregate-items. Therefore, the performative structure has two scenes (in addition to the Start and End scenes), one for each subtask. There are three roles (x, y, z) involved in that performative structure, a coordinator to be played by the agent applying the task-decomposer, and as many operators as subtasks. In the example there are two operators, one (y) participating in the Elaborate-items (EI) scene, and another (z) participating in the Aggregate-Items (AI) scene. Notice that the coordinator (x) should be the same in both scenes, it enters first the EI scene, and can enter the AI scene only after finishing the EI scene.

The task-based performative structure used for specifying the operational description of a task decomposer keeps the decoupling of tasks and capabilities. This approach, which aims at maximizing capability reuse, leads to the use of scenes based on two generic roles. And the low granularity of the two-role scenes used as the building blocks of a performative structure, together

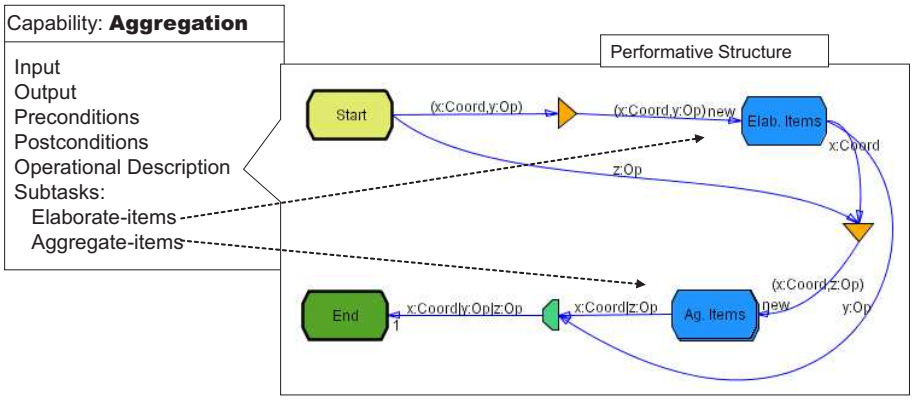


Figure 5.16: Task-decomposer operational description

with the existence of the-facto standards for one-to-one interaction (e.g. the FIPA Request-Inform protocol), are two extra features supporting the goal of maximizing reuse.

Using performative structures to describe the operational description of a capability enables parallelism and provides an abstract view of the coordination required for Teamwork, which can be sensibly used to improve the Team Formation process by producing more robust teams and improving the overall performance of the team.

We have explained the ORCAS model of the Cooperative Problem Solving process, and the ORCAS team-model. Moreover we have introduced a formalism to describe the communication and the operational aspects required to turn the Knowledge Modelling Ontology into a full-fledged ACDL. Now we are in position to focus on the two operational stages of the CPS process: Team Formation and Teamwork.

5.5 Team Formation

Team Formation is the process of selecting a group of agents that have complimentary skills to achieve a global goal (the team goal), and providing team members with the information required to achieve the global goal in a cooperative way.

Team Formation in ORCAS is guided by a task configuration. Since a task-configuration specifies the tasks to be solved and the capabilities to apply, the number of possible teams is reduced, making Team Formation feasible in practice. In large systems, team selection may involve an exponential number of possible team combinations, and a blow-out in the number of interactions required to select the members of a team [Kinny et al., 1992]. ORCAS addresses this problem by introducing the Knowledge Configuration process before Team Formation in the Cooperative Problem-Solving process [Wooldridge and Jennings, 1999].

Our model of the Team Formation process considers three activities, namely: *task allocation*, *team selection* and *team instruction*.

- During the *task allocation* process candidate agents are obtained for each task, according to the requirements of a task-configuration;
- next, during the *team selection* process, some agents are selected for each specific team-role, while other agents are kept in reserve for the case of agent failure;
- finally, during the *team instruction* process, agents involved in the team formation process are informed about the result of the team configuration stage: the team roles they have to play, and the social knowledge required to cooperate with other team-members during the Teamwork process.

Later, in Chapter 6 a particular agent infrastructure supporting the ORCAS framework is presented. This infrastructure provides the services required from both providers and requesters of capabilities to form customized teams of agent on-demand. The approach there is to include institutional agents acting as middle-agents with the capabilities required to configure tasks and coordinate the Team Formation and the Teamwork processes. Since there are a lot of strategies and algorithms for team formation and agent coalition formation, we want to explain Team Formation from a more conceptual point of view, focusing on the inputs, the outputs and the requirements of the Team Formation process, rather than describing how the process is carried on in the ORCAS implemented infrastructure, addressed in Chapter 6.

5.5.1 Task allocation

Task allocation is the process of selecting a group of candidate agents to form a team, such that their aggregated competence satisfies the requirements of the problem at hand. This process follows the task decomposition structure defined by a task-configuration to know which are the tasks to be allocated, and looks for candidate agents that are suitable to solve each task.

Task-allocation can be performed by a middle agent, facilitated by it, or distributed among several agents. Since the ORCAS implemented infrastructure aims at minimizing agent requirements and facilitate teamwork, the ORCAS infrastructure relies upon middle agents to support both providers and requesters of capabilities during the CPS process. Specifically, the ORCAS infrastructure provides a kind of broker called Team-Broker, which is able to form teams on-demand, according to a task-configuration.

A team is defined as a hierarchy of team-roles derived from a task-configuration. Each subtask within a task configuration defines a team-role that should be played by someone, and more specifically, at least one agent must commit to each team-role in order to complete the task allocation process. Candidate agents are those than in addition to be equipped with the capability required for a team-role, accepts to play that team-role.

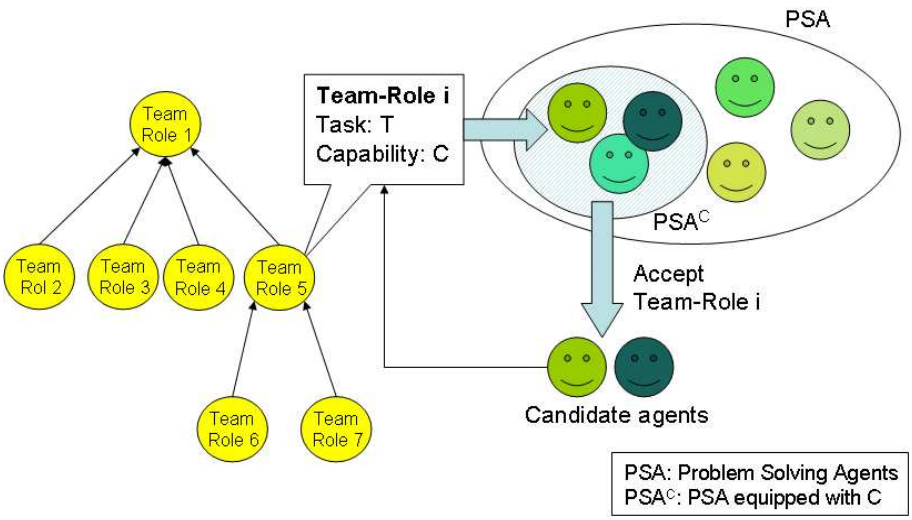


Figure 5.17: Task allocation

Figure 5.17 shows the task-allocation process as a filtering process. From the Problem-Solving Agents (PSA) available, only those equipped with a team-role's capability are potential candidate. At the end, only the agents accepting to play a team-role with the specified requirements (the capability and the domain-knowledge to use in order to solve the team-role's task) become candidate agents. The Task Allocation process proceeds until there are candidate agents for all the team-roles composing a team. In the example, there are three agents equipped with the required capability (C), which become potential candidates. In the end, only two agents have accepted to play that team-role.

Although we avoid establishing a model of commitment based on mental attitudes (joint intentions, joint commitment) some model of agency is required to implement the CPS process. Herein we will rely on a weak notion of agency, specifically, an implicit model of commitment will be assumed. From this approach, commitment is implicit in the act of accepting a team-role proposal; in other words, when an agent accepts a team-role proposal during the team selection process, one assumes that the agent is committing to achieve the corresponding task using the selected capability, as specified in the task-configuration.

In the ORCAS implementation of an agent infrastructure supporting the Team Formation process, task allocation is a responsibility of the Team Broker role. Although individual agents may be self-interested, we assume that there is a global interest for the team to offer together a good service, and thus, some service provided by the infrastructure to select appropriate agents appears as an interesting feature, for instance, selecting the agents with lower workload, or agents with a cheaper cost, depending on the preferences specified for the

problem at hand.

In the ORCAS infrastructure, a Team-Broker role is defined that is able to obtain candidate agents by using a task configuration as the source to generate team-roles, and sending team-role proposals to potential candidate agents. The agents receiving a team-role proposal can autonomously decide to agree, to refuse the proposal, or to make a counter-proposal. The Team-Broker role has to wait until all the available agents have answered or a time-out is reached. Candidate agents will be used to select the members of the team, as explained in the next subsection.

5.5.2 Team selection

Team selection is the process of selecting a set of team members from the collection of candidate agents obtained during the task-allocation process.

The result of the Team Selection process in ORCAS is a *team-configuration*, which results of instantiating the abstract team-roles of a team model with specific agents selected from candidate agents. A team-configuration is obtained by selecting a group of agents, and optionally some reserve agents, for each team-role. All agents selected and kept in reserve for some team-role become team-members.

A team-configuration is complete when all the team-roles composing a team (Definition 5.4) are complete:

$$Complete(Team(\pi^0, Conf(\kappa))) \Longleftrightarrow \forall \pi \in Team(\pi^0, Conf(\kappa)) : \pi_{A_S} \neq \emptyset$$

where π_{A_S} is the set of agents selected to play team-role π. Otherwise $Team(\pi, Conf(\kappa))$ is partial. In other words, a team-configuration is complete when there are agents selected to play all the team-roles composing a team.

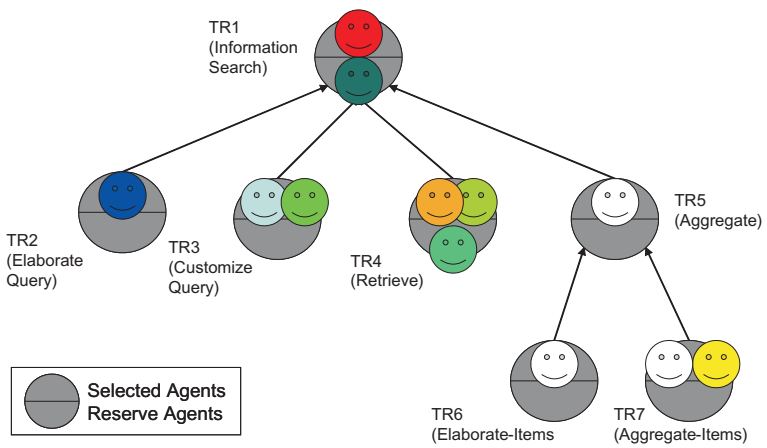


Figure 5.18: Example of Team-configuration

Figure 5.18 shows an example of a team-configuration. In particular, this example shows a team-configuration for a team that has to solve the Information-search task. Some team-roles are assigned a single agent (TR1, TR2, TR5, TR6), while other team-roles have several agents selected (TR3, TR4 and TR7), besides some team-roles include reserve agents (TR1, TR4). In some cases, the agent selected to apply a task-decomposer has to delegate all the subtasks to other agents (TR1), while in other cases the task-decomposer agent is assigned some (or all) of their own subtasks (TR5).

A second goal of team selection is to decide the scenes to be used between agents requiring some communication. Since ORCAS teams are hierarchically organized, every interaction occurring during the Teamwork process involves an agent playing a task-decomposer team-role and acting as the coordinator, and one agent playing a subordinated team-role (there is one subordinated team-role for each subtask). The coordinator is responsible for decomposing the problem and delegating the subtasks to the agents selected for a subordinated team-role, distributing data to other agents, receiving back the results, and performing intermediate data processing between subtasks.

Team selection in the ORCAS Operational Framework can use different selection criteria and can be carried on according to different strategies and interaction protocols. There are just a few requisites imposed by the ORCAS operational framework to be satisfied by the Team selection process:

- Selecting at least one agent for each team-role, except when there are alternative team-roles, since then, an agent selected for any of the alternative team-roles is enough.
- Selecting a scene for each team-role, such that the scene is shared by both the agent willing to play the coordinator team-role, and the agent willing to play the operator team-role. In other words, two agents must share a common scene in order to communicate.
- Optionally, non-selected candidate agents can be kept as reserve agents for the case the selected ones could not achieve their tasks.

In the ORCAS infrastructure, described in Chapter 6, the team selection process is performed through an auction-like protocol driven by the Team-Broker role.

Figure 5.19 sums up the process of choosing the communication scenes to be used for each team-role. First of all, one or more agents are selected to play every team-role. Secondly, a single scene is selected for every team-role. These scenes are selected from the scenes shared by both the agent willing to act as the operator, and the agent willing to act as the coordinator. In the example, agent A is selected to play Team-Role 7, which is subordinated to Team-Role 5), allocated to agent B. Therefore, some moment during the problem solving process A and B must communicate following certain scene, B acting as coordinator and A acting as operator. Such a scene specifying the communication between both team-roles is selected from the intersection of the communication scenes supported by

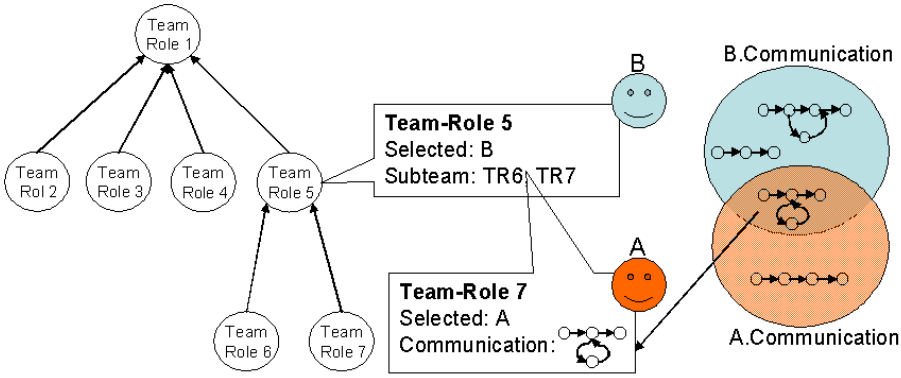


Figure 5.19: Choosing communication scenes during the team selection process

both agents, and is assigned to the communication slot of Team-role 7, the one playing the operator role.

Summarizing, the result of the Team selection process is a *team-configuration*, a hierarchical structure of interrelated team-roles complying with a task-configuration. Each team-role within a team-configuration defines the following elements (Definition 5.1): a task to be achieved; a capability to achieve the task; a set of agents selected to play the team-role; a set of agents to keep in reserve; a communication scene specifying the interaction protocol; optionally, the domain knowledge to be used by the capability; and finally, if the capability is a task-decomposer, a team-role must include a subteam feature describing the team-roles subordinated to this team-role (see §5.3.1), and specified as team-components (Definition 5.2).

Although there are multiple strategies allowed by the ORCAS framework to assign agents to team-roles (to allocate tasks to agents), there are some general considerations to take into account.

On the one hand, agents can play more than one role in a team and thus they can be selected to occupy several positions. Consequently, an agent playing both a task-decomposer team-role and some of the subordinated team-roles can reduce communication costs by performing the tasks assigned to both the task-decomposer and the subordinated roles. However, agents may be selected taking into account their workload, in order to balance the global performance of the MAS, that can be performing multiple tasks in parallel.

On the other hand, the ORCAS approach aims at exploiting the information provided by the operational description of a task-decomposer to know which are the task dependencies. This information can be used when selecting the agents that will play each team-role, trying to increase the possibility of success and improving the overall team performance. We consider four types of task relationships that can be very useful during the Team Formation process, namely sequences, choices, parallelism and multiple-instances.

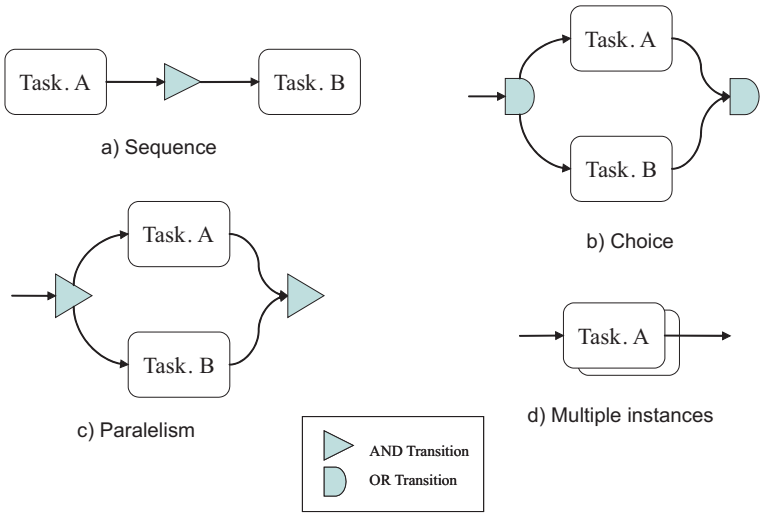


Figure 5.20: Representing control-flow in a performative structure

In order to characterize the task relationships involving a group of alternative (choices) or parallel tasks, we will use the term "fade-in" transition to refer to an initial transition agents are forced to traverse before performing any of the tasks in the group, and the term "fade-out"is used to denote the transition agents must reach in the end, after performing any of the tasks. The fade-in transition has outgoing edges going to the alternative/parallel tasks, and the fade-out transition has incoming edges coming from all these tasks. We describe below the four types of relationships considered within the ORCAS framework, and discuss briefly the way they can be used to improve the Team Formation process:

- *Sequences* (Figure 5.20.a) are defined among tasks than should be solved one after another. Usually, two tasks should be performed sequentially when there is some data dependencies between them (the output of one task is the input of another task). Tasks to be performed sequentially have an AND transition between them. Sequential tasks do not allow parallelism, therefore it is not advantageous to select different agents to solve them. The agent selection criteria for sequential tasks is independent from the other tasks.

- *Choices* (Figure 5.20.b) are used to define alternative tasks to choose from. An agent faced with a set of alternative tasks can choose any of them to progress through the performative structure. Choices are specified by a set of alternative tasks preceded by an OR (fade-in) transition and followed by an OR (fade-out) transition too. There are an outgoing edge from the fade-in OR transition to each alternative task, thus an agent traversing that transition can move to any of the following tasks by choosing one

among the outgoing edges. The OR transition after the alternative tasks allow an agent having performed some task to progress through it without waiting for agents performing other tasks. In general, one single agent is sufficient to allocate a set of alternative tasks, since only one task is strictly to proceed further. However, some times may be useful to try several alternative tasks and then retain the result of only one task, for instance, the first finished task. Therefore, when there are candidate agents for several alternative tasks, it might be interesting (though not necessary) to select agents suitable for several alternative tasks.

- *Parallelism* (Figure 5.20.b) means that several tasks must be performed in parallel. Parallel tasks are represented between an AND fade-in transition, an AND fade-out transition, and several outgoing edges connecting the fade-in transition to every task. All the tasks between the fade-in and fade-out must be performed in parallel. The fade-in (AND) forces agents traversing it to follow all the outgoing edges. The fade-out transition (AND) ensures that all the tasks to be performed in parallel have finished to allow agents to proceed further. In order to exploit parallelism the Team-Formation process should select different agents to perform parallel tasks.

- *Multiple instances* (Figure 5.20.b) means that a task can be performed multiple times in parallel. Multiple instances are represented by overlapped tasks within an ORCAS performative structure. For example, the Retrieve task appears within the Meta-search task-decomposer's operational description as allowing multiple-instances. This is due to the fact that Retrieve takes a single query as input, while the previous task (Customize-Query) produces a set of queries. For this reason the Retrieve task must be repeated for each query produced by the Customize-query task. Since multiple instances of a task may be solved in parallel, it could be beneficial to assign several agents to that task.

5.5.3 Team instruction

Team Instruction is the process of informing each team member about all the information they need to play their team-roles during the Teamwork process. Team-roles have been defined in Section §5.3, and we have stated that team-role structures are used during the Team Formation process not only to send proposals to join a team during the task allocation process, but also to instruct team members on the tasks they have to solve, the capabilities to apply, and all the information required to communicate with other team-members. Specifically, we use team-roles to inform an agent willing to apply a task-decomposer on which tasks to be delegated to other agents, to whom, and which communication scenes to use. A useful distinction is established distinguish between team-roles assigned to a skill, and team-role assigned to a task-decomposer.

Figure 5.21 shows an example of a team-role endowed with a skill. This team-role is associated to the task Elaborate-query, and specifies that the ca-

pability Query-elaboration-with-thesaurus, which is a skill, should be applied for solving Elaborate-query. Furthermore, that skill has to use the domain knowledge characterized by the MeSH thesaurus domain model.

Team-Role	
Task	Elaborate-Query
Team-Id	Team-23
Role-Id	Role-2
Capability	Query-elaboration-with-thesaurus
Domain-Models	MeSH-Thesaurus

Figure 5.21: Team-role example for a skill

Figure 5.22 shows an example of a team-role endowed with a task-decomposer. This team-role corresponds to the Information-search task, which is bound to the Meta-search capability. This capability is a task-decomposer introducing four subtasks: Elaborate-query, Customize-query, Retrieve and Aggregate. Therefore, the subteam for this capability has four team-components, one per subtask. Each team-component in the sub-team identifies both the agents selected to solve one of the subtasks, the agents to keep in reserve for each subtask, together with the task and the identifier of the team-role associated to that task. For instance, in Figure 5.22, the Red agent is selected to play the role assigned to the Elaborate-query task, while the Green agent is kept in reserve. However, sometimes it is desirable to allocate the same task to several agents. In that example there are two agents selected for the Retrieve task, Red and Blue, because that task may be executed multiple times while applying a meta-search capability: the task Retrieve takes a single query as input, while other tasks that are executed previously (Elaborate-query and Customize-query) usually output several queries; thus the task Retrieve has to be performed once for each query. Since these multiple executions may be carried over in parallel, it can be beneficial to distribute the multiple instances of the task among different agents.

The information provided by a team-role to a team-member agent during the team instruction process is used in the following way: an agent being requested to perform a task is provided with a team-identifier and a team-role identifier, so as to allow that agent to retrieve the information about the requested team-role from his local knowledge base. First of all, the agent checks whether is it committed to that team-role or not. Following, the agent finds out whether the team-role's capability is a skill or a task-decomposer. If the team-role capability is a skill, then the agent applies that skill and sends the result back to the requesting agent. Otherwise the capability is a task-decomposer, and the agent has to check the team-role subteam to find out whether it has to delegate some subtask to another agent. This information is provided by team-components; if a team-component has selected agents different of himself, then the corresponding subtask must be delegated to that agent. The communication scenes to be used

Team-Role				
Task	Information-Search			
Team-Id	Team-23			
Role-Id	Role-1			
Capability	Meta-Search			
Subteam				
	Subtask	*Role-Id*	*Selected*	*Reserve*
	Elaborate-quey	Role-2	Red	Green
	Customize-query	Role-3	Red	Yellow
	Retrieve	Role-4	Red, Blue	
	Aggregate	Role-5	Blue	Cyan

Figure 5.22: Team-role example for a task-decomposer

are also specified by the team-components.

To sum up, the team instruction process provides team-members with all the information they require activity to cooperate with other team-members during the Teamwork process.

5.6 The Teamwork process

The Teamwork process comprehends all the activities a team must carry out to solve a problem. In ORCAS the Teamwork process aims at solving a problem according to its requirements. In order to do that, during the Team Formation process a group of agents have joined a team by committing to some team-roles. The resulting team is customized for the problem at hand, since it is based on a task-configuration satisfying the stated problem requirements. Team members have been instructed on the tasks they have to solve (one task for each team-role), and on the capabilities they must apply to solve each task.

Whilst the Knowledge-Configuration process operates over static information describing agent capabilities from an abstract view, the Teamwork process has to take into account the dynamic nature of the environment. A team solving a problem in a real environment has to deal with events and conditions occurring at runtime, which may difficult the achievement of the team goals, e.g. excessive workload, agent failures or communication problems.

A team is composed of a task-coordinator playing a task-decomposer team-role, and a sub-team. The sub-team coordinator has to apply a task-decomposer capability to achieve its goals. The operational description of a task-decomposer describes the control flow over subtasks as a performative structure (§5.4.3).

The performative structure describing a task-decomposer's operational description is specified as a network of interrelated scenes, one for each subtask. Each scene requires at least two agents to be carried out: a *coordinator*, which is played by the agent assigned to the task-decomposer team-role, and one agent assigned to a subordinated team-role and playing the *operator* role. The for-

mal scenes describing the communication required to achieve each task have been decided during the team selection process (§5.5.2). Consequently, in order to apply a task-decomposer, the coordinator agent has to initiate the different scenes while following the performative structure.

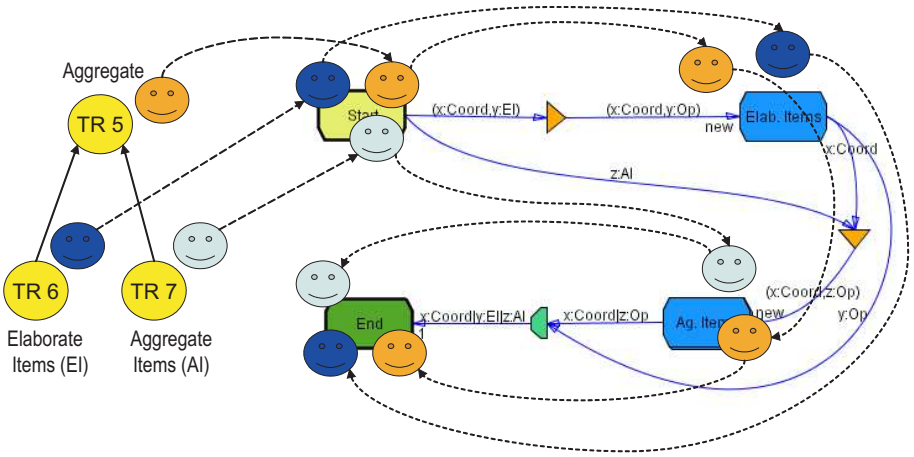


Figure 5.23: Teamwork model for a task-decomposer

Figure 5.23 shows the role flow policy through the performative structure describing the operational description of a task decomposer. Specifically, this figure shows the operational description of the Aggregation capability, which is bound to Team-role 5 (TR5), together with the paths to be followed by the agents selected for this team-role and the subordinated team-roles: TR6, TR7. The Aggregation capability is a task-decomposer introducing two subtasks: Elaborate-items (TR6) and Aggregate-items (TR7). We note A^{TRi} as the agent selected to play Team-Role i. All the agents begin at the Start scene, and then:

1. A^{TR5} and A^{TR6} move from the Start scene to the Elab.Items (EI) scene; A^{TR5} adopts the Coordinator role (x:Coord), while A^{TR6} takes the Operator role (y:Op). Both roles are required to perform the scene, so the edge going from the first transition to this scene is a conjunction of them (x:Coord, y:Op).

2. A^{TR7} moves from the Start scene to the Ag.Items(AI) by taking the Operator role (z:Op) and waiting A^{TR5} at the AND transition placed between the Elab.Items scene and the Ag.Items scene. A^{TR5} gets to that AND transition after playing the Coordinator role at the Elab.Items scene ($x : Coord$). The AND transition forces the incoming agents ($x : Coord, y : Op$) to synchronize before proceeding to the Ag.Items scene. As we can see in the picture, after crossing that transition, both agents continue playing their previous roles, A^{TR5} as Coordinator, and A^{TR7} as Operator.

3. Agents acting as operators can leave the performative structure just af-

ter finishing the scenes they have participated in. In particular, A^{TR6} and A^{TR7} can move to the End scene through an OR transition from the Elab.Items and the Ag.Items scenes respectively. However, the Coordinator role cannot abandon the performative structure until all the scenes have finished.

Teamwork is guided by a task-decomposer performative structure. Since some subordinated team-roles are also assigned to task-decomposers, new performative structures should be initiated when an agent assigned to a task-decomposer is requested by his coordinator. Therefore, the Teamwork process can be modelled as a nested structure of performative structures. There is one performative structure for each task-decomposer, starting from the top team-role.

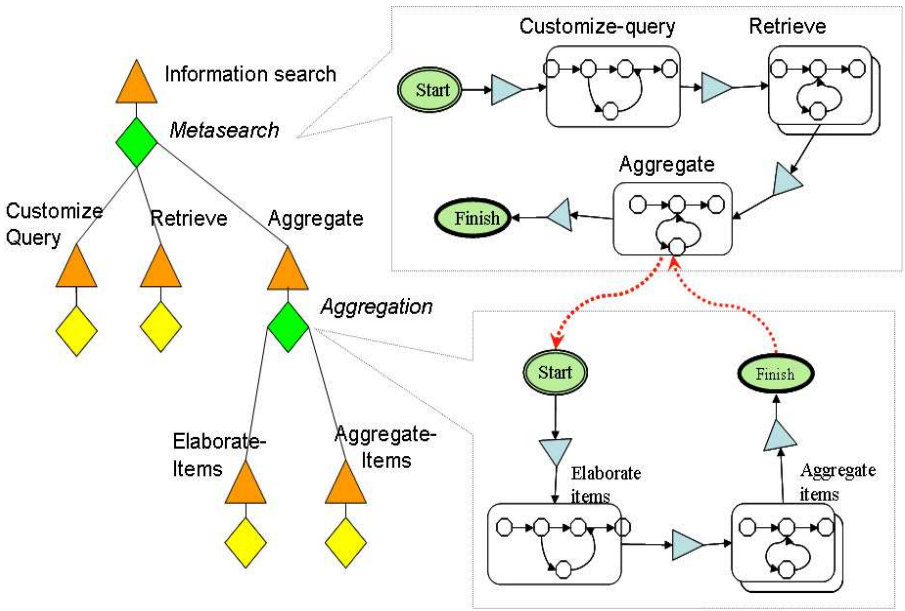


Figure 5.24: Teamwork model for a team

Figure 5.24 sums up the specification of the teamwork activity as a nested structure of performative structures. Notice that there is a performative structure for each task-decomposer in the task-configuration and one scene describes the interaction protocol required to delegate one task to another agent. The performative structures indicate which roles are required to play each scene, and the dependencies among scenes, e.g. some scenes must be finished before starting another scene, other scenes can be performed in parallel, or an scene can be instantiated multiple times in parallel. Moreover, the agent acting as the coordinator within a performative structure is as well holding the information about the team mates assigned to its subtask, specified within its team-role subteam

(Definitions 5.1 and 5.2). The coordinator agent can initiate each scene at the right moment by contacting the agent or agents assigned to the corresponding team-role and following the selected scene. The agents playing some team-role must wait until a new scene is initiated by the coordinator. Moreover, a performative structure can include choice points that give the coordinator alternative paths to follow. It is a design decision to develop complex task-decomposers with many alternatives or to build many simpler decomposers with few alternatives or no alternatives at all.

The teamwork process follows the hierarchical structure of the task-configuration, decomposing a task into subtasks when there is a task-decomposer, and delegating some subtasks to other team members. The teamwork process starts with the team-leader (the agent assigned to the root task in the task-configuration) having to apply a task-decomposer. The team-leader starts the team-work process by following the performative structure that specifies the operational description of its task-decomposer. The team-leader engages in conversations with their subordinated agents in order to delegate them the subtasks specified in its task-decomposers. A task is delegated by following the scene specified by a team-role, providing the input for the data to the selected agent (as indicated by the selected agents feature of the team-role), and receiving the result for that task.

When a subordinated agent has to apply a task-decomposer itself, it does the same that the team-leader: delegates subtasks to selected agents and wait for the results, aggregate the results when opportune and send the global result to its own coordinator. The process of applying a task-decomposer follows the performative structure. Since some of the subtasks may be bound to task-decomposers too, a new performative structure must be carried over for each task-decomposer. The first performative structure (the one initiated by the team-leader) cannot finish until the new performative structures are finished. Therefore, performative structures are nested, a performative structure cannot finish until any performative structure under it finishes.

When a subtask is allocated to the same agent applying the task-decomposer, the scene associated to that task in the performative structure is skipped, since there is no need for communication. Instead, the agent solves the subtask himself by applying the required capability, realize that communicating with himself has no sense.

When a subtask is allocated to another agent, the agent applying a task-decomposer initiates the scene specified in the team-component associated to that subtask by sending the first message. For instance, if the scene specifies a Request-Inform protocol, then the coordinator sends a request with the following information:

1. a team identifier;

2. a team-role identifier;

3. the input-data required by the subtasks associated to that team-role

All this information is included within an object of the sort Perform, as defined in the Teamwork ontology 5.6.

5.7 Extensions of the Operational Framework

The Cooperative Problem Solving process already presented has some limitations that arise when addressing runtime time dependencies among tasks and dynamic events altering the expected outcome of the Teamwork activity.

On the one hand, two common perturbations in the CPS process came from agent failures (i.e. an agent is unable to achieve a task) and communication errors (e.g. a message does not get to its destination). Since Team Formation can assign reserve agents, some times it is still feasible to resume the CPS process after an agent failure, without reconfiguration, by requesting reserve agents to play the associated team-role. However, sometimes there are no reserve agents to perform the unfinished tasks, and then a reconfiguration mechanism is required to look for agents equipped with alternative capabilities (i.e. other capabilities suitable for the task at hand and compatible with the problem requirements if possible).

On the other hand, some tasks may need or may benefit from information obtained at runtime in order to be configured. A task is configured by selecting a capability suitable for it, binding the capability to the task, and recursively configuring the subtasks of the capability when it is a task-decomposer. Sometimes, the selection of one capability or another may be improved or requires some information obtained at runtime. Therefore, the Knowledge Configuration process should be delayed for those tasks until the required information is obtained. In order to configure those tasks, we allow a capability to produce information to be used by the Knowledge Configuration process, such as a new precondition stated to be true, a new postcondition to be achieved, or a new domain model characterizing new domain knowledge (some capabilities may generate domain knowledge).

A more flexible CPS process is required to deal with such situations; therefore, we introduce some extensions to the CPS process that consist in different ways of interleaving the Knowledge Configuration, the Team Formation and the Teamwork processes.

5.7.1 Interleaving Teamwork, Knowledge Configuration and Team Formation

We consider three strategies that interleave Teamwork, Knowledge Configuration and Team Formation, namely *Reconfiguration*, *Delayed-Configuration* and *Lazy Configuration* .

Reconfiguration occurs when a task bound to a capability cannot be achieved by neither the selected nor the reserve agents allocated to it. The purpose of reconfiguring a task is to find another capability suitable for that task.

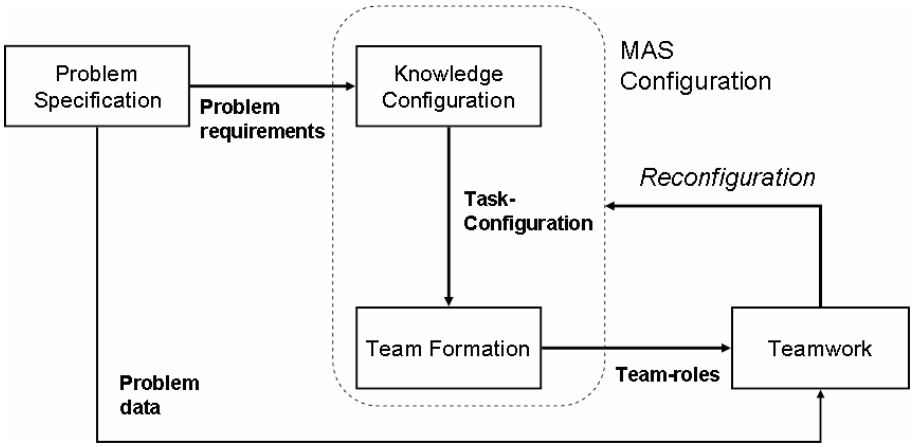


Figure 5.25: Extended model of the Cooperative Problem Solving process

Figure 5.25 shows an extended model of the Cooperative Problem Solving process capturing the notion of Reconfiguration. If the new capability bound to the task is a task-decomposer, then their subtasks must be further configured. A capability satisfying the global problem requirements is required, though a partial satisfaction criteria can be used instead to allow the Knowledge Configuration to succeed even when a fully satisfactory condition is unreachable. If the new capability bound to the task is a skill, then the reconfiguration ends there, otherwise the capability is a task-decomposer requiring a recursive configuration of the new subtasks.

The *Delayed-Configuration* strategy is used to hold the configuration of some task up until some event happens or some information is obtained.

Figure 5.26 shows a capability that performs a Propose-Critique-Modify method over the Information Search task, by decomposing it into three subtasks: P-Search (Propose-Search), C-Search (Critique-Search) and M-Search (Modify Search). The M-Search task is decomposed by the task-decomposer Modify-metasearch into four subtasks; the first of these tasks, Adapt-query, can be solved by two skills: Query-generalization and Query-specialization. The selection of one skill out of the two former skills depends on the result of the P-Search task: on the one hand, if P-Search obtains too many results then the capability skill Query-specialization is preferred; on the other hand, if the are slender results that are considered not enough then Query-generalization is better. Otherwise, the number of results is considered adequate and the task M-Search is discarded without further configuration.

In order to use the information obtained in runtime at the Knowledge Configuration process, we have specified the Query-generalization and Query-specialization capabilities as having different, incompatible postconditions: Query-generalization includes the postcondition Generalize-query, while Query-specialization includes the postcondition Specialize-query. There is only one capa-

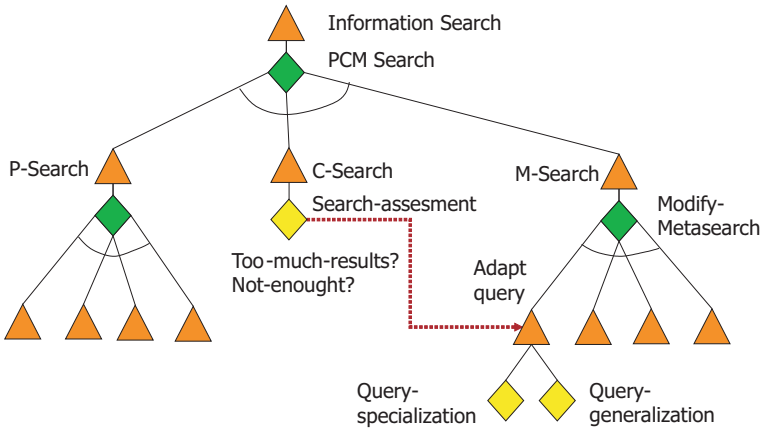


Figure 5.26: Propose-Critique-Modify Search

bility suitable for the task C-Search, Search-assessment. Search-assessment brings about one of the two former formulae according to whether there are too many results for the query, or there are not enough results as to be useful. Furthermore, if the number of results obtained for the task P-Search is considered adequate, Search-assessment outputs a different formula expressing that condition, so as to allow the coordinator realize the task M-Search can be omitted.

The coordinator of a task specified as requiring a Delayed-Configuration, must be aware of the conditions in order to to interrupt and resume the Teamwork process when required, and perform the necessary actions to assure a new Knowledge Configuration process is initiated using the new conditions bring about in runtime. Similarly, the coordinator of a task that cannot be achieved using the current configuration must initiate a new Knowledge Configuration process in order to find an alternative task-configuration.

In both cases, after a new Knowledge Configuration process is over a new Team Formation process is required to allocate the new tasks to a new team. The resulting team acts as a subteam of the original team, taking responsibility of the team-role corresponding to the task that has just been configured.

In addition, other situations characterized by very dynamic environments may benefit of a systematic *delayed configuration* strategy, because it has no sense to completely configure a task in advance in such a situation, or tasks will probably require a reconfiguration too often. We call this strategy *Lazy Configuration*. The idea of the *Lazy Configuration* strategy is to avoid configuration whilst possible, configuring a task just when it is required to be solved. Lazy configuration is used to handle very dynamic environments that made profitable to gradually configure a task whereas the Teamwork process progresses. In our implementation of Lazy Configuration, the Knowledge Configuration process operates by configuring every time just one level of the task decomposition structure of a task-configuration. Using this strategy, when a task-decomposer

is bound to a task, each subtask is bound a capability, but the newly introduced task-decomposers are not further expanded into new tasks. Once a task is configured one-level deep, the Teamwork process runs until a new task-decomposer has to be applied that introduces some new tasks to be configured, and then the Knowledge Configuration process should be performed again following the Lazy Configuration strategy, and so forth for each new task-decomposer going to be applied. A variation of the Lazy Configuration strategy is the introduction of variable deep levels during the Knowledge Configuration stage.

An interesting possibility is to perform the different activities of the CPS process simultaneously. After starting a Knowledge Configuration process using the Lazy Configuration strategy, perform Team Formation and Teamwork next, but instead of stopping the Knowledge Configuration process after starting Team Formation, continue with the Knowledge Configuration while possible, running in parallel with the Team Formation and the Teamwork processes, in such a way that when the Teamwork has to solve a task that is bound to a task-decomposer, the task will be already configured. We call this strategy *far-sighted* strategy.

The consequence of introducing these variations is a greater flexibility of the original model of the CPS process to handle different kind of scenarios, though the more flexible the configuration strategy is, the more communication activity is required.

5.7.2 Operational scenarios: dimensions and some prototypical scenarios

This subsection will draft a future work discussion on dimensions that may constrain the application of the ORCAS framework, and some typical scenarios that can fit well in the ORCAS framework.

The **autonomy dimension** deals with the degree of autonomy possessed by agents. Very autonomous agents are designed for distributed control approaches in which agents keep local control during most of the problem solving process. In addition to decide the commitment to a team-role, autonomous agents may prefer to decide by themselves the plans to use for achieving the goals of a team-role, i.e. configure a task by himself, and deciding when and whom to delegate a task. On the other hand, agents may prefer to delegate some activities of the cooperative process to specialized agents, like brokers and matchmaker agents. Team Formation and Teamwork strategies may adopt a wide range of strategies according to the degree agents keep local control during the problem solving process. While a distributed control approach is more appropriate for dynamic environments and real-time applications, a more centralized control is better suited for a service-oriented model of the CPS process. The ORCAS Operational Framework is neutral about the autonomy dimension. We are not specifying here which agents are responsible for configuring a task or forming a team. Later, in Chapter 6, an infrastructure for developing and deploying agents according to this framework is presented. This infrastructure is based on the electronic

institutions approach, which adopts and external view and is based in the idea of standardized patterns of behavior called agent roles.

Distributed control approaches may be implemented over this infrastructure by equipping problem solving agents with the capabilities and the permissions required to play institutional roles: the Knowledge-Broker, responsible for configuring a task during the Knowledge Configuration process, and the Team-Broker, responsible for selecting and instructing agents during the Team-Formation process. Although we have initially defined the Knowledge Configuration process and the Team Formation process as being entirely completed before moving to the following stage, we have presented also some extensions of the Operational Framework that allow to interleave all the processes involved in the CPS process: Knowledge Configuration, Team Formation and Teamwork. Specifically, the combined use of the Lazy Configuration strategy during the Knowledge Configuration process and the adoption of the Knowledge-Broker and Team-Broker roles by problem solving agents covers most of the existing distributed approaches to team formation with autonomous agents.

The ORCAS implementation of an agent infrastructure (Chapter 6) aims to support agents developed by third parties to partake in the CPS process, and thus there are institutional agents equipped with the reasoning abilities required to configure tasks and form teams. However, it does not imply that control is centralized in a classical sense, it rather means that requesters and providers are mediated by institutional agents facilitating their work. From a service oriented or a component-based software development (CBSD) approach (e.g. "off-the-shelf" components), providers (problem solving agents) may be interested on cooperating with other agents and relying on institutional agents to carry out the Knowledge Configuration process and drive the Team Formation process. The point is that in these approaches the global interest represented by the problem requirements or the user preferences is usually favored against the interest of individual capability providers. The institutional agents included in the ORCAS infrastructure bring an added value to both the requesters and the providers of capabilities, freeing them of complex and computationally intensive tasks like configuring a task or selecting the members of a team. Therefore, this facility promotes a light-weight approach to agent development, and is oriented towards compositional software development approaches in which applications are composed rather than constructed, by reusing existing components (agent capabilities and domain knowledge).

The **openness dimension** deals with the capacity of integrating external agents and other components (like knowledge repositories, databases and Web services). Open agent societies allow external agents to joint the society on runtime, on a dynamic basis, without having to recompile the system code. Openness is also related to the maintenance, extensibility and adaptiveness of a system, since a system can be modified or extended by incorporating or eliminating agents. Open agent systems are designed to facilitate the integration of heterogenous agents provided by different developers. However, the greater the

openness, the greater the complexity.

A main topic concerning this subject is the management of ontology mismatches, that is to say, allowing agents to use different ontologies and handling the mapping required to translate concepts from one ontology to another. Connectors between components can be introduced to implement ontology mappings; like the *bridges* proposed in the UPML software architecture [Fensel et al., 1999]. In the current implementation of ORCAS we are using a common ontology to avoid ontology mismatching and focus on other aspects such as the coordination of agents. Nonetheless, the ORCAS Abstract Architecture is well suited to introduce such kind of components, indeed, because of the conceptual decoupling of tasks, capabilities and domain models. In ORCAS connectors could be inserted between capabilities and domain-models, and between tasks and capabilities as well. The use of connectors in ORCAS would has two dimensions: a knowledge-level specification, which allows to match components specified with different ontologies; and the implemented counterpart, which allows semantically (or syntactically) heterogeneous agents to interoperate during the Teamwork process.

To sum up, both the configuration and the coordination of agent teams can be distributed using the same basic model of the Cooperative Problem Solving process. Current research on coalition formation algorithms use distributed algorithms to deal with the combinatorial nature of this class of problems. Optimal anytime coalition structure generation algorithms has been devised [Shehory and Kraus, 1998, Sandholm et al., 1998, Larson and Sandholm, 2000]. Some minor modifications of the ORCAS framework are required to support a distributed approach to the Knowledge Configuration and the Team Formation processes. In a distributed scenario, agents should consider both task-dependencies and problem requirements when configuring a task. Since independent agents have a partial view of the problem, cooperation and coordination with other agents to look for an optimal global solution would be required.

We consider now some prototypical scenarios that may fit into the ORCAS framework for the Cooperative Problem Solving process: the *Agent Factory* model, the *Service Orchestration* model, and the *Contractual Agent Society*.

The *Agent Factory* model is based on a notion of production factories, and has been proposed as a design pattern by the Object Oriented Programming (OOP) community. The idea is to assemble existing components to build customized solutions. This model fits well with the "Off-the-Shelf" Components approach to software development and shares some similarities with the Software Configuration community. Concerning agents, the Agent Factory model can be applied to build agents or teams on-demand, according to some existing requirements, rather than forming a team from a set of pre-existing agents. We consider two approaches to the Agent Factory model:

- building and assembling team-specific agents from elementary components;
- instantiating and coordinating generic agent types.

In the first approach, agents can be as complex as necessary in order to

minimize the number of agents participating in a team, so as to reduce communication overhead. In the second approach, agents are pre-built, though they can be somehow parameterizable. While the first approach favors Teamwork, the second one speeds up Team Formation.

The *Service Orchestration* model refers to the activities required to select, compose and execute several Web services to achieve a global task. The Service Orchestration model proposed by the Semantic Web Services approach has similar goals and shares many similarities with the ORCAS framework when comparing services against capabilities. Semantic Web services can be conceptually described as capabilities in ORCAS. The DAML-S ontology defines the following aspects of a service: a *profile* that brings the information needed by service-seeking agents to determine whether the service meets its needs; an *process model* on how does the service works, which should facilitate service composition and monitoring; and the *grounding*, which specifies the way to invoke and interact with a service:

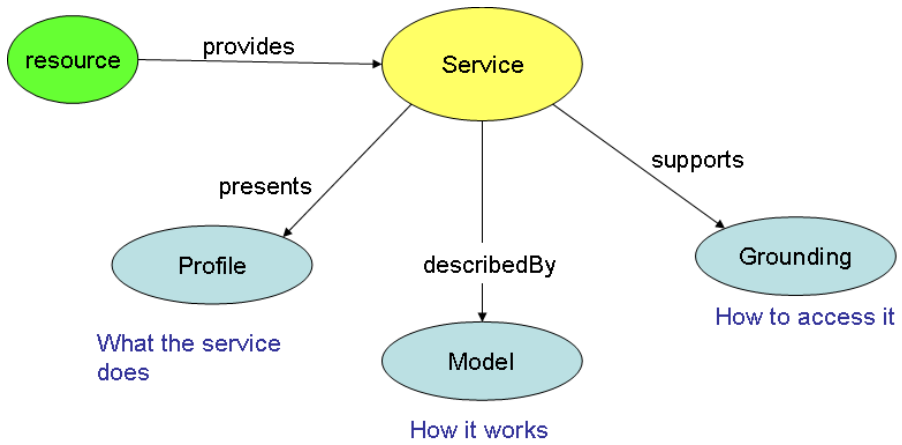


Figure 5.27: Service description according to DAML-S

Figure 5.27 shows the main elements of a service description according to the DAML-S ontology [The DAML-S Consortium, 2001]. All these aspects of a service have an equivalent in ORCAS:

- The aspects playing the role of the profile are provided by the knowledge-level description of capabilities.
- The purpose of *communication* of a capability is equivalent to the grounding of a service.
- The *operational-description* addresses the same features covered by the process model of a service.

Table 8.1 summarizes the relation between the features characterizing a capability in ORCAS, the features proposed to describe agent-enabled semantic

DAML-S	Agent activities	ORCAS ACDL
Profile	Discovering (matchmaking)	Inputs, outputs and competence
Grounding	Invocation and Execution	Communication
Operational model	Composition and Interoperation	Subtasks and operational description

Table 5.1: Comparison of the ORCAS ACDL and DAML-S

Web services in the DAML-S ontology [The DAML-S Consortium, 2001], and the kind of activities these features are required for [Bansal and Vidal, 2003, Bryson et al., 2002, Park et al., 1998, Payne et al., 2001].

Due to these similarities between the Semantic Web Services (SWS) approach and the ORCAS framework, we think the ORCAS framework could be easily adapted to work upon SWSs as the providers of capabilities. Moreover, the use of agent wrappers over SWSs will allow the full integration of SWSs and agents within the ORCAS infrastructure.

The proposed *Contractual Agent Society* (CAS) model [Dellarocas, 2000] relies on a contractual agreement strategy for trusting agent interaction. Both providers and requesters of a service must agree upon the conditions associated to the provision of a service. Both the requesters and the providers have to comply with the conditions established by the contract, moreover, the contract specifies also the consequences of violating those conditions. The idea of using contracts fits well with the electronic institutions approach, in fact, we are considering as future work the introduction of what we call "terms of commitment" (Chapter 8), as a mechanism to agree upon by team members when accepting a team-role. This feature remains for the future work. e-Commerce applications like supply chains, auctions and e-markets are all based on contracts, therefore these applications are good candidates to apply the ORCAS framework.

5.8 Conclusions

Starting with a general set of requirements, we have stated the operational information to attach to the knowledge-level description of capabilities in order to become a full-fledged Agent Capability Description Language: the communication requirements of any capability, and the operational description of task decomposer capabilities. We have proposed electronic institutions concepts to specify such aspects of a capability. Specifically, communication requirements are specified as scenes —interaction protocols— and dialogic frameworks. Moreover, performative structures are used to specify the operational description of a task-decomposer.

The ORCAS infrastructure has been designed and implemented as an electronic institution. Actually, the ORCAS infrastructure can be seen as a meta institution where dynamic problem-solving institutions are configured on-the-fly,

according to stated problem requirements. Some elements from the electronic institution formalism have been incorporated as components of the ORCAS Agent Capability Description Language. These elements —scenes, dialogic frameworks and performative structures— are defined for each capability, and are composed during the configuration of an agent team, which is done in two steps: first a task-configuration is obtained that specifies the competence required for a team to comply with stated problem requirements, and second, a team of agents is formed according to the task-configuration and ensuring that all the agents involved can interoperate by sharing a common institutional framework: communication language, ontologies, and interaction protocols (scenes).

Chapter 6

The Institutional Framework

This chapter describes an open agent infrastructure to develop and deploy MAS according to the ORCAS framework. This infrastructure is an electronic institution where problem solving agents meet to form teams and solve problems on-demand, according to the ORCAS model of the Cooperative Problem-Solving process.

6.1 Introduction

This chapter presents a particular implementation of the Knowledge Modelling Framework and the Operational Framework as an electronic institution, which is called the ORCAS e-Institution. The ORCAS e-Institution is an infrastructure for developing and deploying cooperative Multi-Agent Systems that supports both providers and requesters of capabilities along the different stages of the CPS process.

We have already presented a model to configure a MAS at two layers: the knowledge layer, called Knowledge Configuration, and the operational layer, called Team Formation. Moreover, we have presented a framework for the execution stage of the CPS process, that we call Teamwork.

This chapter describes an open agent infrastructure designed to use the ORCAS KMF as and Agent Capability Description Language, according to the two layers configuration model. The goal of this infrastructure is to allow the development and deployment of open, reusable and configurable Multi-Agent Systems:

- *Open:* agents can be created in multiple programming languages and interface with existing legacy systems.
- *Reusable:* tasks and capabilities are declared in a domain-independent way using its own domain-independent ontologies

- *Configurable on-demand:* components (capabilities and domain knowledge) are selected according to problem requirements and user preferences

During the rest of the chapter, and after a general overview of the ORCAS e-Institution in §6.2, we are going to review its dialogic framework §6.3, performative structure §6.4, and the communication scenes §6.5, in this order.

6.2 Overview of the ORCAS e-Institution

The ORCAS e-Institution acts as a mediation service for both clients and providers of capabilities. Both requesters and providers of capabilities are ruled by well defined interaction protocols (scenes), and mediated by institutional agents that reason about application tasks, agent capabilities and problem requirements using the ORCAS KMF as the Agent Capability Description Language. The ORCAS e-Institution is used to configure a MAS for a particular problem, which involves all the stages of the Cooperative Problem Solving process as described in the Operational Framework: Knowledge Configuration, Team Formation and Teamwork. The configuration of the MAS includes the Knowledge Configuration and the Team Formation process, and the result is a customized team of agents that is tailored to solve a specific problem according to its requirements. ORCAS teams are created and instructed to solve specific problems by obtaining a knowledge level configuration (a task-configuration) in terms of goals to achieve (represented as tasks), the competence (the capabilities) required by team members to achieve those goals, and the domain knowledge (satisfying the assumptions of the selected capabilities).

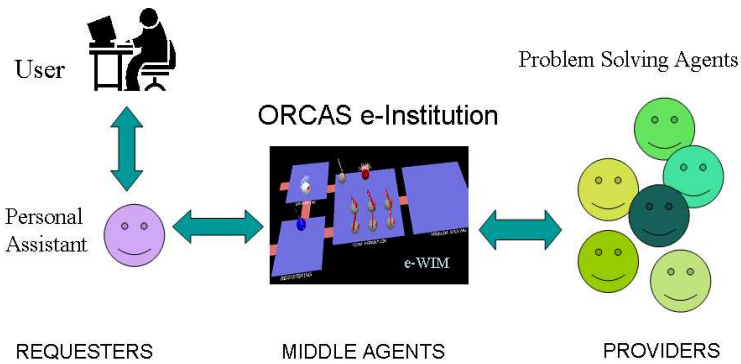


Figure 6.1: The ORCAS e-Institution as a mediation service between requesters and providers of capabilities

The ORCAS institutional framework is based on a client-server architecture extended with the notion of middle-agents [Decker et al., 1997b]. This model involves three kind of agents: providers, requesters and middle agents.

- *Providers:* agents providing specific capabilities to solve application tasks. In our architecture we call them Problem-Solving Agents (PSA).
- *Requester:* human or software agents requesting to solve a problem. We have included a class of agent called Personal Assistant to act on behalf of a human user. The Personal Assistant frees users of knowing the technical details needed to interact with other agents.
- *Middle agents:* agents mediating between requesters and providers. These agents are responsible for finding providers and instructing them to solve a problem. We consider three classes of middle agents: *librarians*, *knowledge-brokers* and *team-brokers*. *Librarians* act like a "yellow pages" service. They are dynamic repositories of agent capabilities, providing the link between the knowledge layer (task-configurations) and the operational layer (agent teams). *Knowledge-brokers* are able to obtain configurations of the MAS in a declarative manner, according to a problem specification. *Team-brokers* deal with the operationalization of a configuration, which consist in forming a team of problem solving agents with the capabilities required by a task-configuration.

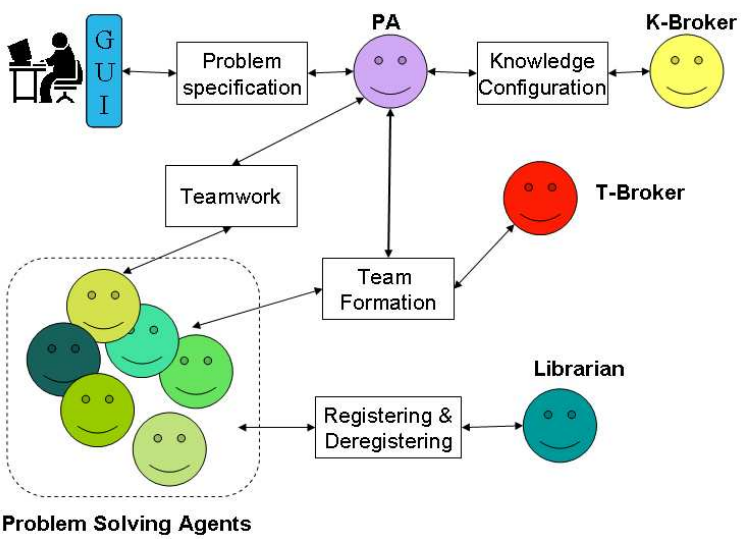


Figure 6.2: ORCAS e-Institution: main agent roles and activities where they are involved

Figure 6.2 shows the main agent roles that can be played by agents participating in the ORCAS e-Institution: Personal Assistant, Problem Solving Agent, Knowledge-Broker, Team-Broker, and Librarian. Bidirectional arrows represent the communication scenes where agent participate to carry out the different stages of the CPS process: Registering/Deregistering, Problem specification, Knowledge Configuration, Team Formation and Teamwork.

The rest of the chapter explains in detail the ORCAS e-Institution, using the ORCAS KMF as the ACDL and supporting all the stages of the ORCAS model of the CPS process (Knowledge Configuration, Team Formation and Teamwork). Since many elements of the electronic institutions formalism used within this chapter are described in Chapter 5 (The Operational Framework), we will refer the reader to the appropriate sections within that chapter when introducing each element of the ORCAS e-Institution.

6.3 Dialogic Framework

The dialogic framework specifies the ontological elements and communication language (ACL) employed during agent interactions. A dialogic framework can be defined either globally, for the entire e-institution, or using one dialogic framework per scene.

There are two kinds of agent roles in an electronic institution: *external* roles, than can be played by external agents, and *internal*, institutional roles. In our case, we consider as external roles the ones played by both requesters and providers of capabilities, namely the Personal Assistant (PA) and the Problem-Solving Agent (PSA) roles. However, middle agents are defined as internal roles, belonging to the institution: Librarian, Knowledge-Broker and Team-Broker (T-Broker).

There are five main agent roles in the ORCAS e-institution, namely Personal Assistant (PA), Librarian, Knowledge-Broker, Team-Broker, Problem-Solving Agent (PSA), plus two subroles of the PSA role: Coordinator and Operator.

1. *Personal assistant (PA):* An agent acting on behalf of a human user. This agent is responsible for mediating between the user request and the services offered by the application. The PA is able to specify problems in terms understood by the Knowledge-Broker, that is to say using the ORCAS KMF meta-ontology and Feature Terms as the object language. Furthermore, the PA is able to interact with the Team-Broker during the Team Formation process, and to start the Teamwork activity once the team is formed.

2. *Librarian:* This agent holds the knowledge descriptions of the reusable components: tasks, capabilities, domain-models and ontologies. The library can be dynamically updated or extended with new component descriptions by following a registering/deregistering procedure. Therefore new agents can enter the system by registering their capabilities to the Librarian using the ORCAS KMF as ACDL. The Librarian agent is a dynamic repository of ORCAS KMF components, allowing other agents or humans to query about them. Hence, the Librarian can be used as a "yellow pages" service, just keeping an up-to-date record of the association between ORCAS components and the agents that registered them. In the ORCAS e-institution, the Librarian is queried by the Knowledge-Broker to

know which are the components available to the Knowledge-Configuration process.

3. *Knowledge-Broker:* The purpose of the Knowledge-Broker is to configure an application for a user (represented by the PA) requesting to solve some problem. The Knowledge-Broker uses the specification of a problem as an input to generate a task-configuration: a structure of ORCAS components matching the problem specification, using tasks, capabilities and domain-models. The Knowledge-Broker is thus the responsible for performing the Knowledge Configuration process.

4. *Team-Broker:* The purpose of the Team-Broker is to operationalize a task-configuration by forming a team of problem solving agents. A team is a group of agents committed to solve a problem together, according to a task-configuration. A team is formed by finding and selecting agents with the required capabilities, and instructing them to cooperate in solving a problem together.

5. *Problem-Solving Agent* (PSA): This role is adopted by the agents willing to provide some capabilities. Problem-Solving Agents can join or leave the system dynamically, just registering or deregistering their capabilities to the Librarian. This is a simple way to make the Librarian aware of the capabilities available in the system at any moment.

Figure 6.3 shows the agent roles defined by the ORCAS e-Institution. Notice how they are organized according to the two kinds of agent relationships established by the electronic institutions formalism: SSD (dotted lines), and a partial order relation defined as a subclass relationship ($\succeq$).

Since the Knowledge Broker, the Team Broker and the Librarian roles are institutional roles, they are preempted of being adopted by an agent playing an external —non institutional— role. This SSD policy protects the institution from being used by external agents to favor themselves in detriment of other agents. This constraint reinforces the trust of external agents on the institution.

Moreover, we have defined two subclasses of the PSA role, which can thus be adopted by any agent playing a PSA role: the *Coordinator* and the *Operator*.

- The *Coordinator* role is adopted by an agent playing a task-decomposer team role and having to delegate some subtask to other agents. During the Teamwork process, any agent having to apply task-decomposer must initiate the scene specified each subtasks within its team-role subteam. Meanwhile, the agents assigned to team-roles associated to a subtask adopt the operator role. The coordinator is responsible for distributing problem data and intermediate results among the operator agents, and is responsible for coordinating them. An agent acting as coordinator follows the performative structure that specifies the operational description of its task-decomposer, initiates communication scenes to interoperate with its subordinated agents, and performs any intermediate data processing when required.

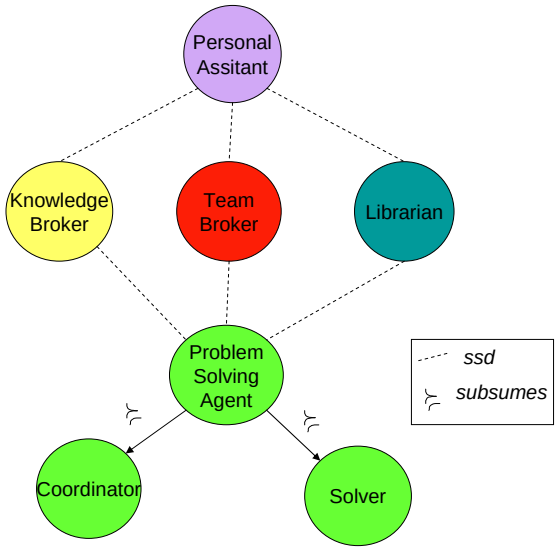


Figure 6.3: ORCAS e-institution roles

- The *Operator* role is adopted by the agents willing to perform a subtask of a task-decomposer. The agent assigned to the task-decomposer adopts the coordinator role and is the initiator of the scenes used to interact with every operator. The operators receive the data they require to solve their tasks from the coordinator, perform the capabilities assigned to their tasks, and send back the results to the coordinator.

Both the Coordinator and the Operator roles are dynamically assigned to PSAs, that is to say, they are not assigned to an agent when it enters the institution, but during the Teamwork process. An agent playing the PSA role switches to either the Coordinator or the Operator roles during the Teamwork stage, according to the following rules: when an agent has to communicate with another agent playing a subordinated role, the first agent adopt the Coordinator role and initiates a scene in which the second agent takes the Operator role; complementarily, if an agent has to communicate with an agent he is subordinated to, he adopt the Operator role on-demand, just after receiving a message from another agent adopting the Coordinator role.

A PSA can be playing both the Coordinator and the Operator roles simultaneously at different scene instances. Since a subtask may itself be bound to a task-decomposer, the agent selected for such a subtask has to act as the Operator with respect to the agent he is subordinated to, but before finishing that interaction, the same agent may partake in other scenes with its subordinated agents as operators, and acting himself as coordinator. Figure 5.18 shows an example

of a team where this condition will happen: the agent selected for Team-role 5 (TR5) is assigned the task Aggregate, which is a subtask subordinated to TR1 (task Information-Search); thus the agent in charge of TR5 has to play the operator role with respect to the agent in charge of TR1. But while engaged in a scene with the agent playing TR1, the agent playing TR5 has to engage a scene acting itself as coordinator, while the agent playing TR7 acts as operator, since TR7 is assigned the task Aggregate-items, which is subordinated to TR5.

The Coordinator and the Operator role are not under a SSD relationship, because an agent can play, as exemplified above, both roles at the same time. As we explain later concerning Teamwork, during the Teamwork scene 6.5.4, team members can engage new problem solving scenes when needed, adopting either the Coordinator or the Operator role according to the subordination relations established by the hierarchical structure of a team-configuration **??**.

```
(define-dialogic-framework ORCAS_e-Institution_df as
    ontology = (KM-Ontology Teamwork-Ontology Brokering-Ontology)
    content-language = NOOS
    illocutionary-particles = (request inform accept refuse)
    external-roles = (PSA PA)
    internal-roles = (Librarian T-Broker K-Broker)
    social-structure ((PA ssd PSA)))
```

Figure 6.4: ORCAS e-institution dialogical framework

Figure 6.4 shows the dialogic framework of the ORCAS e-institution. The ontology contains the vocabulary and the concepts used by agents to communicate when other agents participating in the institution. Any ORCAS e-Institution comprehends at least three ontologies, the Knowledge-Modelling Ontology, the Brokering Ontology (describe later, in §6.5.2), and the Teamwork Ontology, which contain all the concepts required by the different roles of the ORCAS e-Institution to participate in the institution. Moreover, any application of the ORCAS e-Institution adds library specific concepts that should be shared by the agents participating in that application, like the concepts included in the ISA-Ontology (D), which is used by the agent participating in the WIM application (§7). The content language is NOOS [Arcos, 1997], a reflective object-centered representation language designed to support systems integrating knowledge-modelling and learning.

In order to implement the ORCAS framework as an electronic-institution, we have added some concepts to the Teamwork ontology which are required by institutional agents to communicate at the different scenes of the ORCAS e-Institution. These concepts should be understood by both the external and the internal, institutional agents in order to interoperate. These concepts will be introduced as needed when describing the different scenes of the ORCAS e-Institution within this chapter. As we have introduced in Chapter 5, we use elements of e-Institutions to specify the operational description and the commu-

nication requirements of agent capabilities; however, these elements should be distinguished from the ORCAS e-Institution.

On the one hand, the ORCAS infrastructure follows the original electronic institutions formalism ([Esteva et al., 2002b]) in that is is specified as a fixed structure of scenes (a performative structure), which are known beforehand.

On the one hand, the communication supported by an agent over a capability is specified as a scene (§5.4.2), while the the operational description of a task-decomposer is specified as a performative structure (§5.4.3), but these, these elements do not make up an electronic institution as conceived in the referred formalism; instead, they constitute a collection of incomplete performative structures (recall that scenes are not typed) that are completed (by selecting the scene types) and composed dynamically during the Teamwork process following a nested structure. These structures can be seen as special class of electronic institutions that we like to call dynamic e-Institutions, since they are configured on-the-fly, according to the requirements of the problem at hand.

Actually, since the nesting of performative structures occurs within the Teamwork scene in the ORCAS e-Institution, the ORCAS e-Institution can be seen as a meta-institution that defines the environment for the execution of dynamic, throw-away e-Institutions. This idea introduces becomes a new topic we put off as deserving further research (Chapter 8).

6.4 Performative structure

The performative structure of an ORCAS e-institution represents the network of interaction scenes, together with the relationships among scenes, that describe the paths followed by agents playing some role in the institution. The ORCAS performative structure contains four communication scenes, plus the Start and the End scenes, from where agents enter and exit the institution. The main scenes of this institution are the following:

1. *Registering scene:* where a PSA can register its capabilities to the Librarian in order to become available for the CPS process, as well as deregister when leaving the institution. The registering/deregistering process keeps the Librarian with an up-to-date description of the capabilities available at the system at any moment.

2. *Brokering scene:* this scene describes the pattern of interaction needed to obtain a task-configuration by the Knowledge Configuration process. The participants are a PA requesting to find a task-configuration, the Knowledge-Broker responsible for building the task-configuration, and the Librarian, holding the current description of the capabilities available in the system (the library of problem-solving components).

3. *Team Formation scene:* this scene describes the communication protocol for selecting and instructing the members of a team. The agents involved

are the PA, that holds the task-configuration obtained by the Knowledge-Broker at the Brokering scene, the available Problem-Solving Agents, that wait for team-role proposals to join a team, and the Team-Broker, responsible for selecting the team members and instructing them on the way to cooperate and coordinate with other team mates.

4. *Teamwork scene:* finally, once a team of agents has been formed and instructed to cooperate, the team-mates go to this scene to solve the problem in a cooperative way, using the information provided to them during the Team Formation scene by the T-Broker. The agents involved are the PA, holding the input data for the problem at hand, and the selected team members (PSAs).

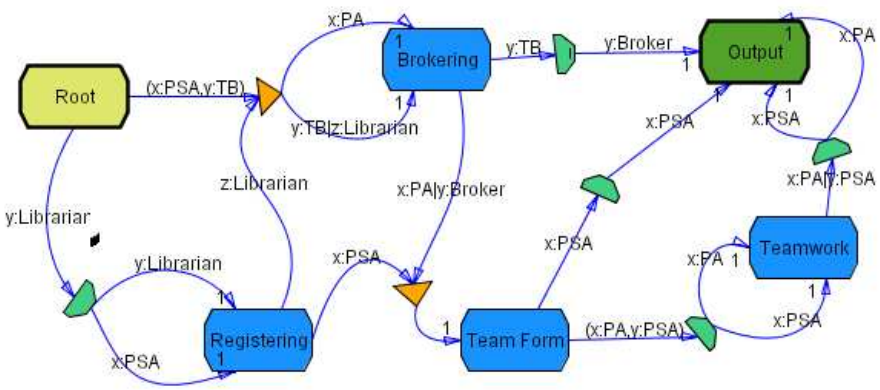


Figure 6.5: Performative structure of the ORCAS e-institution

Figure 6.5 shows the graphical representation of the ORCAS e-Institution performative structure. In addition to the four main scenes (Registering, Brokering, Team Formation and Teamwork) there is a root scene and an output scene as the initial and final scenes respectively. The role-flow policy is represented by the edge labels and transitions between scenes. Notice a PSA has to move first to the Registering scene, and only then can a PSA move to the Team Formation scene to wait for team-role proposals. The PA must start in the Brokering scene to request the K-Broker for a task-configuration satisfying the requirements of a problem; afterwards the PA moves to the Team Formation scene to request the Team-Broker for a new team-configuration. Afterwards, the PA moves to the Teamwork scene to request the team-leader of the recently formed team to solve the problem. Finally, the PA provides the team-leader with the input data for the problem at hand, and waits for the results. The different communication scenes are described in the following section, devoting one subsection for each scene.

6.5 Communication scenes

This section describes the different communication scenes in the performative structure of the ORCAS e-institution: registering and deregistering (§6.5.1), brokering (§6.5.2), Team Formation (§6.5.3) and Teamwork (§6.5.4).

6.5.1 Registering scene

The Registering scene describes the communication activities used to allow the Librarian to be aware of the agents available in the system at any moment, and the capabilities they are equipped with.

Actually, there are two complementary activities, registering and deregistering. On the one hand, PSAs willing to join the ORCAS e-Institution must register their capabilities to the Librarian agent; on the other hand, PSAs willing to exit the institution must inform the Librarian they are leaving, so as to allow the Librarian update the library.

The Registering scene follows a request-inform protocol. When a PSA enters the agent platform, it sends a "register" message with the set of capabilities it is equipped with. The Librarian builds a table with the bindings between capabilities and agents, and keeps it updated by tracking the registering and deregistering activities of PSAs.

Figure 6.6 shows the registering scene specified as a request-inform protocol. The scene starts with a PSA requesting the Librarian to register a set of capabilities (transition 1). The Librarian can then accept (transition 2) or refuse that request. If accepted, the Librarian informs the PSA whether the requested capabilities have been successfully registered (transition 3) or there was some problem (transition 5).

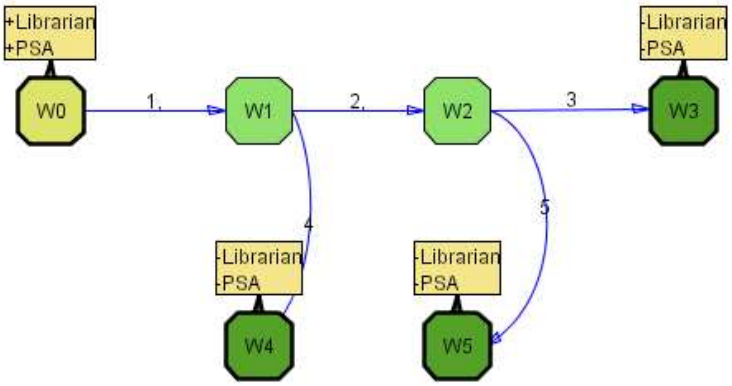
6.5.2 Brokering scene

The purpose of the Brokering scene is to allow the PA to obtain a task-configuration satisfying specific problem requirements. Recall that a task-configuration is obtained through a Knowledge Configuration process (§4.4), which takes a specification of problem requirements and a specification of the components in the library (tasks, capabilities and domain-models) as inputs, and produces a task-configuration as output.

There are three roles participating in the Brokering scene: the Knowledge-Broker, the Personal-Assistant and the Librarian.

The Knowledge-Broker (K-Broker) is the role played by the agent responsible for the Knowledge Configuration process, which is implemented as a search over the space of possible configurations, where the problem requirements are constraints to be satisfied by the configuration.

The Personal Assistant (PA) role represents the user and deals with all the human-computer interaction. The PA is defined as an external role, since only the communication layer of the PA are domain-independent. In general, the PA has a common social layer that defines the acceptable behavior of the PA



1. (W0 W1) request (?x PSA) (?y Librarian) register(?capabilities))
2. (W1 W2) accept (!y Librarian) (!x PSA) register(!capabilities))
3. (W2 W3) inform (!y Librarian) (x! PSA) register(!capabilities))
4. (W1 W4) refuse (! Librarian) (!x PSA) refusal(?reason))
5. (W2 W5) inform (!y Librarian) (!x PSA) failure(?error))

Figure 6.6: Specification of the Registering scene

within the institution, while there is an application specific and even a user specific layer dealing with the particularities of a specific application or user. In the ORCAS implemented e-Institution, the domain-independent layer of the PA is separated from the application specific details, which are implemented separately, as as pseudo-agent that manage the graphical interfaces. There are several interfaces, some of them are general, domain-independent, and oriented towards expert users (e.g. the knowledge engineer), whilst others are application specific, like those included in the WIM application to interact with the end-user (Chapter 7). An example of a domain-independent interface is shown in Figure 4.19, while examples of domain-dependent interfaces are depicted in Figure 7.17 and Figure F.4.

Moreover, the PA role defines just the communication requirements for an agent holding the specification of a problem to be solved so as to interact with the rest of agents in the institution. The basic function of the PA is to mediate between the user and the institution so as to relieve the user of holding any knowledge about how to locate and interoperate with other agents. Rather than including domain-specific knowledge as part of the PA, we preferred to implement a generic, domain-independent PA, and include domain-dependent knowledge as part of the interface (interfaces are implemented as pseudo-agent that can communicate with an agent using the agent communication language). To sum up, the PA brings an added value to the application tasks by providing customization services, and domain-informed support to the decision taking during the Problem Specification process.

The problem specification process is skipped here, since it is performed outside the institution; however, we assume that the PA is holding a complete

specification of a problem to be solved, represented using the ORCAS ACDL and the vocabulary from the application ontology (e.g. the ISA-Ontology in WIM).

The Brokering scene begins when the PA sends a request message to the Knowledge-Broker (K-Broker) containing a specification of problem requirements. The K-Broker is an agent that is able to obtain a task-configuration satisfying specified problem requirements on-demand, out of the component specifications hold by the Librarian.

Once the K-Broker receives a request from the PA, it asks the Librarian to get an updated version of the components registered in the library at the moment, searches a task-configuration satisfying the problem requirements, following one of the three configuration strategies described in §4.4.4.

The K-Broker can ask the Librarian for the entire library at the beginning, or it can ask the Librarian several times, requesting only those components satisfying a matching criteria so as to retrieve only useful specifications. For instance, when going to bind a capability to a task, the K-Broker can ask the Librarian for just those capabilities matching that task.

The concepts required to participate by the PA and the K-Broker to participate in the Brokering scene are included in the Brokering ontology Figure 6.7.

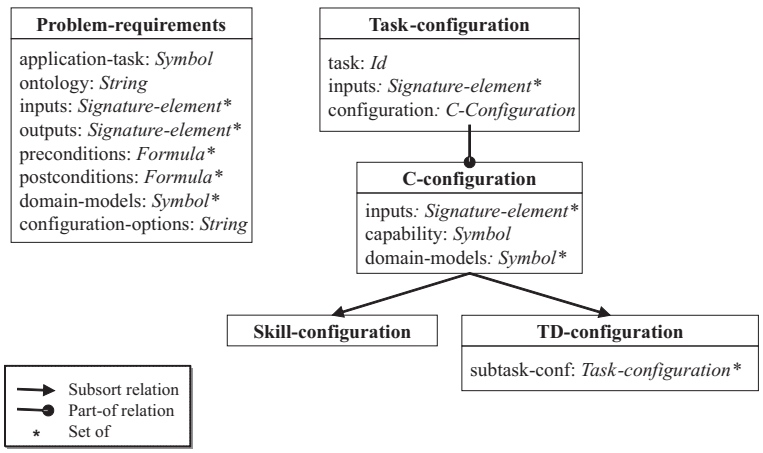


Figure 6.7: Broker Ontology

The input for the Knowledge-Configuration process is a specification of *problem requirements* composed of a) the name of the task to be achieved, b) the pre-conditions that are established to hold, c) the postconditions that have to be hold when the task is achieved, and d) the domain-models that are available for achieving the task. The outcome of the Knowledge-Configuration process is a task-configuration: a tree of triplets containing a task, a capability suitable for that task, and one or more domain models satisfying the knowledge requirements of that capability.

Figure 6.8 shows the interaction protocol for the Brokering scene. The scene starts (w_0) with the PA sending a message to the K-Broker. This message (transition 1) is a request to obtain a task-configuration using the problem specification included in the content of the message. In the next state (w_1), the K-Broker can either asks the Librarian for the entire library (transitions 2, 4), or it can ask for a partial set of tasks or capabilities satisfying some matching criteria (transitions 3, 5). The purpose of these actions are to make the K-Broker work with an updated version of the library during the Knowledge Configuration process, while the use of one or another mode depends on the strategy adopted by the K-Broker. In the first case the K-Broker gets the entire library in one single interaction, while in the second case the K-Broker takes several steps to obtain all the required information, but can retrieve only the information that is strictly necessary to configure a particular task, rather than using the complete library. The first step in the Knowledge Configuration process is to choose which task characterizes better the problem at hand. In order to do that, the K-Broker sends the set of tasks matching the initial problem specification to the PA (w_6), ranking those tasks according to the similarity measure defined in the context of the Case-Based Knowledge-Configuration strategy (§4.5). The PA chooses one task from that set and informs the K-Broker (transition 7) on the selected task and the configuration strategy to use. After receiving the specification of components from the Librarian, the K-Broker starts a Knowledge Configuration process over those specifications. If the K-Broker succeeds obtaining a task-configuration satisfying the requirements, it sends a inform message containing the resulting task-configuration to the PA (tr. 8), and the scene ends.

Notice that the process may fail at several points, causing the scene to end without obtaining a task-configuration. The Brokering scene (Figure 6.8) includes a second final state that is reached either when the K-Broker cannot find a task satisfying the requirements of the problem (tr. 10), or when the K-Broker cannot obtain a task-configuration (tr. 9).

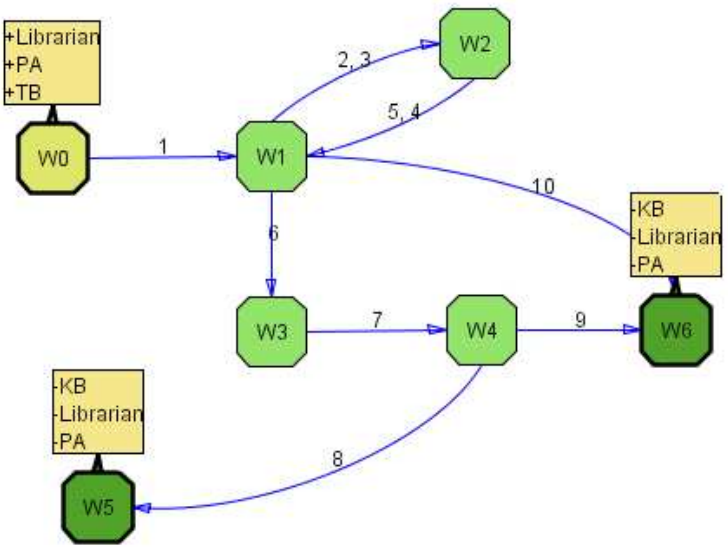
The Knowledge-Configuration process has been described in Section §4.4. This process is performed by the K-Broker during the Brokering scene, at state w_4. The Knowledge Configuration process is performed by the K-Broker as a state-space search in the space of partial configurations (Section §4.4.5).

6.5.3 Team Formation scene

The Team Formation scene describes the communication required to form a team of agents that is able to solve a problem according to a task-configuration. During the Team Formation scene, a team structure composed of team-roles is build according to a task-configuration, and a group of agents is selected and instructed to play every team-role.

We have already described the Team Formation process as having three stages (§5.5): task allocation, team selection and team instruction.

During the task allocation stage, candidate agents are obtained for every team-role. Next, team selection decides the team members to play each team-role out of candidate agents, and keeps other candidate agents in reserve for the



1. (W0 W1) request (?x PA) (?y TB) configure(Problem-Specification))
2. (W1 W2) request (!y TB) (?z Lib) retrieve-component(?specification))
3. (W1 W2) request (!y KB) (?z Lib) retrieve-library)
4. (W2 W1) inform (!z Lib) (!y KB) found(?components))
5. (W2 W1) inform (!y Lib) (!z KB) nothing-found)
6. (W1 W3) request (!KB null) (!PA null) choose-task(?task-list)) ( )))
7. (W3 W4) inform (!x A) (!y KB) configure(?task ?mode)) ( )))
8. (W4 W5) inform (!y KB) (!x PA) configured(?task-configuration))
9. (W4 W6) failure (!y KB) (!z PA) partial-configuration(?task-configuration)) ( )))
10. (W1 W6) failure (!y KB) (!z PA) no-task-found)

Figure 6.8: Specification of the Brokering scene

case of failures during Teamwork. Finally, the agents involved in the process are informed on the result of the team selection process, and the agents selected are instructed on the way they should coordinate with other agents during the Teamwork process.

The Team-Broker (T-Broker) is the responsible of guiding the Team Formation process; it mediates between the PA willing to solve a problem and the PSAs providing their capabilities and waiting for requests to join a team. The T-Broker is able to reason about the component specifications and task-configurations, but it is specialized in forming teams, and has specific knowledge about operational descriptions that are useful to improve the team selection process (and so the performance of Teamwork). Section **??** describes the constructs of an operational description that can be used for that purpose: sequences, choices, parallelism, choices and multiple-instances (Figure 5.20).

First, the Team-Broker obtains candidate agents willing to play some team-role by sending team-role proposals and accounting for the agents accepting. Agents accepting a team-role proposal are considered as committing (by dialogue) to play that team-role, and are consequently expected to try to achieve the associated tasks when required during the Teamwork process. The Team-Broker will analyze the task-configuration to know which tasks should be solved by the team and which capabilities are required to solve each task. Using that information the Team-Broker can generate the team-model, one team-role for each task in the task-configuration, and propose those team-roles to agents willing to join the team.

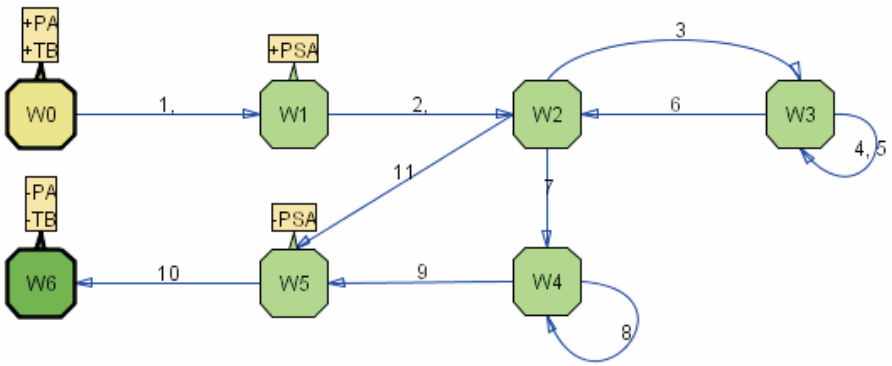
Problem-Solving Agents (PSA) can decide autonomously whether to accept or to refuse a team-role proposal. The Team-Broker waits until all the available agents have answered or a time out is reached, then, the Team-Broker decides among alternative agents for the same task which ones to select as members of the team.

Different algorithms and strategies can be used within the Team Formation scene to allocate tasks to candidate agents. It is possible for instance to select agents after each team-role proposal. Another strategy is first to obtain candidate agents for all the team-roles, and then to select agents for all the team-roles. Moreover, the infrastructure presented here is suitable for a wide range of selection strategies that can be embodied within the Team-Broker agent. The team selection strategy belongs to the Team-Broker internal decision-making strategies, and not to the institutional framework.

Figure 6.9 shows the specification of the Team Formation scene. There are three roles involved: the PA, the Team-Broker and the Problem Solving Agents.

The Team Formation scene starts at w_0 with the PA requesting the T-Broker to form a new team (transition 1) given a task-configuration. Next, the T-Broker initiates the task allocation activity by informing available agents that a new team formation process is going to start (tr. 2).

The task-allocation and team selection processes follow an auction-like approach similar to the *contract-net* protocol [Smith, 1940]: team-roles are proposed to PSA agents willing to join the team (transitions 3); agents have then a



1. (W0 W1) request (?x PA) (?y TB) team-formation(?task-configuration))
2. (W1 W2) inform (!y TB) (all PSA) start-team-formation(?team-id))
3. (W2 W3) request (!y TB) (all PSA) commit(?team-role))
4. (W3 W3) accept (?z PSA) (!y TB) join-team(!team-role))
5. (W3 W3) refuse (?z PSA) (!y TB) join-team(!team-role))
6. (W3 W2) inform (!y TB) (all PSA) time-out(!team-role))
7. (W2 W4) inform (!y TB) (all PSA) start-team-configuration(?team-id))
8. (W4 W4) inform (!y TB) (?z PSA) commit(?team-role))
9. (W4 W5) inform (!y TB) (all PSA) finish-team-configuration(?team-id))
10. (W5 W6) inform (!y TB) (all PSA) finish-team-formation(?team-id))
11. (W2 W5) inform (!y TB) (all PSA) team-failure(?team-id))

Figure 6.9: Team Formation scene

limited period of time to accept (tr. 4) of refuse the proposal (tr. 6) before the time-out is reached (tr. 6).

If there are no candidate agents for all the team-roles, the team can not be formed at all. After performing several attempts without succeeding, the T-Broker informs participating agents of a team-failure (tr. 11) in order to call off the Team Formation scene and discard the ongoing team.

If there are candidate agents for all the team-roles, the task-allocation process succeeds and the T-Broker announces the beginning of the team selection process (tr. 7). During the team selection process, the T-Broker has to select the agents to play each team-role, and the agents to keep in reserve, using the information provided by both the operational description of task-decomposers (containing information about paralellism) and any application specific criteria considered. After being selected to play some team-role (w_4), team-members are informed on the team-roles they are assigned to and have to commit to (tr. 8). After finishing the team selection process, the T-Broker informs participating agents the team-selection is finished successfully (tr. 9).

As described in §5.3.1, team-roles are used to inform team-members about all they need to carry out a task within the team: the task to be solved, the capability to apply, the knowledge to use, and optionally, if the capability is a task decomposer, the information required to delegate subtasks to another team-members.

After selecting the agents to play every team-role, the identifier of the team and the information required to communicate with the team-leader (the top level team-role) are sent to the PA (tr. 10), and the scene ends. The PA does not require a complete team description because the information on each specific team-role has been submitted to each team member during the Team Formation scene.

6.5.4 Teamwork scene

This section describes the process of solving a problem by a team of problem solving agents (PSAs), once the team has been formed and instructed during the Team Formation stage, in such a way that the specified problem requirements are met.

The already introduced scenes (Registering, Brokering and Team Formation) are single scenes. However, the Teamwork scene is not specified by a single scene, though there is an initial scene that is used by the PA to request the team-leader to begin teamwork, provide the team-leader with the data for the problem and wait for the results. Nonetheless, the teamwork activity is not reduced to this initial scene, but will follow a nested structure of performative structures, one for each task-decomposer team-role (§5.6). Moreover, the scenes to be used within these performative structures are not predefined, but have been selected during the Team Formation scene from the set of communication scenes supported by the two agents involved in any task, one playing the Coordinator role, and another one playing the Operator role (§5.4.2).

The Teamwork process is initiated by the PA once the Team Formation has succeeded. The Teamwork scene defines just the initial scene of the Teamwork process, involving two agent roles, the PA and the PSA roles. There is a single agent playing the PA role, while all the team-members play the PSA role. This scene is used by the PA to send the problem data to the team-leader and get the final result back.

The Teamwork follows a request-inform protocol; the PA sends a "request" message to a PSA playing the team-leader team-role, (the responsible for the root task within the corresponding task-configuration). This message contains a team-identifier, a team-role identifier corresponding to the team-leader, and the input data for the problem at hand.

The team-role identifier is required by the team-leader to check up whether it is committed to that team-role, and to retrieve the information associated to that team-role so as to carry out the task associated to it. Since a PSA can participate in different teams at the same time, a unique team identifier is also required to avoid ambiguity.

When a PSA receives a request from the PA, the PSA checks whether it is committed to the team-role specified in the request. If the target PSA founds that team-role stored in its local memory, it accepts the request, or refuses the request in the opposite case.

Next, the PSA checks the type of capability assigned to that team-role in order to figure whether it is a skill or a capability. According to the type of capability assigned to a team-role a PSA can face two situations:

If the capability assigned to the team-leader's team-role is a skill, the PSA has just to apply that skill over the input data and give back the result to the PA. In this situation, the team is composed of only one team-role. Therefore, the Teamwork activity involves only one scene and two agents, one agent playing the PA role, and another agent playing PSA role.

Otherwise, the capability is a task-decomposer, and the PSA has to consider delegating some subtasks to other agents. This information is provided by the team-role's subteam, which specifies the agents selected for each subtask, as well as the scene to be used to communicate with those agents. In this situation, a team is composed of several team-roles, and the Teamwork activity involves one or more performative structures describing the task decomposition control flow, and several scenes to be played, one for each subtask to be delegated to another agent. Recall that there is one performative structure (an operational description) for each task-decomposer capability assigned to a team-role. The idea is that the teamwork activity can be modelled by a electronic institution whose primitive elements are assembled on-runtime, during the team-formation process. Therefore, the teamwork activity follows the performative structure of that institution, expanding to a new performative structure each time a new task-decomposer is applied. Agents adopt the coordinator and operator roles as required, according to whether they are attending a request from another agent (operator) or they are applying a task-decomposer and have to delegate some subtask to other agents (coordinator).

As team members applying skills finish their tasks, the results are sent back to the coordinators, which are responsible for obtaining the result for his task using the result of the many subtasks, and propagating his own result back to his respective coordinators, and so on, until the team-leader obtains the final result and sends it to the PA.

Chapter 7

Application: The Web Information Mediator

This chapter describes the Web Information Mediator (WIM), a specific MAS application that uses the ORCAS framework and is implemented over the the ORCAS e-Institution. WIM is a configurable meta-search application to find bibliographical references in the Internet. WIM is based in a collection of domain-independent cooperative information agents, while the current application domain is medicine. This chapter describes this application, and some extra work performed concerning the interoperability of the ORCAS e-Institution.

7.1 Introduction

The process of information search and aggregation on the Internet is essentially a process of *mediation* between the goal of the user and the information resources that are distributed and heterogeneous. Therefore, an application dealing with such information processing tasks can be seen as a Web Information Mediator (WIM). The overall goal of WIM is to provide a mediation service for information search tasks of a professional user in some domain. A mediator is an agent that offers an added value to the information accessed in other sources [Wiederhold, 1992]. Typical services offered by a mediator include selection of information sources, information retrieval, ranking and fusion of information from different sources.

WIM has been developed and specified according to the ORCAS framework, and it is deployed over an ORCAS e-Institution (implemented over the NOOS Agent Platform). WIM is a configurable application according to Definition 4.15, and as such it has the following properties:

- WIM functionality is hold by a library of tasks and capabilities provided

by agents and specified in a domain independent manner. The application domain is characterized by a collection of domain models characterizing the domain knowledge. However, WIM adds like wrappers to the ORCAS e-Institution architecture, which are used to agentify external information sources.

- Problems are specified as a collection of problem requirements and problem data. A specification of problem requirements includes the name of the task to be solved, domain models characterizing specific domain knowledge, preconditions stated to hold, and postconditions to be satisfied after solving the problem.
- There is a clear separation between the problem solving agents, which are the providers of the capabilities, and the domain knowledge, which is defined independently. The domain models characterizing the application domain are specified for any particular problem as part of the problem requirements.
- WIM applications are configured on-the-fly for a particular request by decomposing the problem task into subtasks, binding capabilities to tasks, and verifying that the assumptions of the capabilities are verified by the domain models specified in the problem requirements.
- WIM on-the-fly applications are designed, operationalized and executed according to ORCAS model of the CPS process (S5.2), which is made up of four subprocesses: Problem Specification, a Knowledge-Configuration, Team Formation and Teamwork.

In few words, WIM is a configurable application that exploits the ORCAS framework to customize agent teams on-demand, according to stated problem requirements. The separation of tasks and capabilities from the domain is an architectural pattern that aims at maximizing the reuse of agent capabilities. The matching relations allow to verify whether a set of domain models characterizing the application domain satisfies the knowledge assumptions of a capability. As a result, WIM agents can be reused over new application domains by specifying the new domain knowledge as a collection of domain-models, and building appropriate ontology matchings when there is an ontological mismatch between the concepts used to specify the new domain and the concepts defined in the WIM ontology.

The WIM mediation process can be seen as a sequence of adaptation steps: to begin with, WIM adapts the user request (a consultation) to generate domain queries, using knowledge from the specific application domain, like medicine; secondly, WIM agents have to adapt the queries expressed in terms of the application domain to the features of specific information sources; lastly, the information retrieved from several queries and different sources is adapted again to obtain a global result to the user request.

An important distinction in WIM is established between a *domain query* and a *source-query*. On the one hand, a domain-query is a description of the type of

information to search from an abstract, conceptual viewpoint, and is expressed in terms of application domain. On the other hand, a source-query is used to specify a query to a specific information source, in terms of the search strategies and the filters allowed by that particular source.

A second consideration is the paradigmatic distinction between the concept of "relevance" in classical information retrieval (IR), and the richer conceptualizations currently in use for intelligent information agents [Carbonell, 2000]. The canonical concept of relevance in IR is a test where the results of a query by a retrieval engine are compared to a gold standard provided by a human expert that assesses false positives and false negatives. The problem of that approach is that *relevance* is independent from the purpose of the specific user in posing a query. Therefore, current research also tries to establish a *utility measure* that is task-dependent: retrieved items are evaluated to determine in which degree satisfy the intended purpose expressed in the user's query [Carbonell, 2000].

WIM takes into account both considerations: on the one hand, WIM allows the user to specify an information search problem in terms of a domain query, and then it uses domain knowledge to transform the domain query into source specific queries; on the other hand, the result of applying each query can be interpreted according to a notion of *utility* to the task and not only of relevance with respect to the content of the query.

Modern information systems should manage or have access to large amounts of information and computing services. The different system components can conduct computation concurrently, communicating and cooperating to achieve a common goal. These systems have been called *Cooperative Information Systems* [International Foundation on Cooperative Information Systems, 1994]. A major goal of this field is to develop and build information systems from reusable software components. This goal can be achieved by assembling information services on-demand from a collection of networked legacy applications and information sources. WIM is a specialization of that general class of systems in the field of MAS, for it is built by connecting the capabilities provided by a society of agents to some domain knowledge specified as a collection of domain-models. WIM is designed as a configurable MAS according to the ORCAS framework, and has been implemented as a society of information agents participating in an ORCAS e-Institution (developed upon the *Noos Agent Platform*). The WIM application is configured by selecting and configuring the capabilities brought by the WIM information agents, and linking them to a medical domain knowledge (medicine in general, and Evidence Based Medicine in particular). The configuration of an information task (a task-configuration) is operationalized by forming a team of agents with the capabilities encompassed by the configuration, and imposing team members a commitment to that configuration. WIM is configured on-the-fly for each request to solve a problem, and the configuration is built according to the kind of problem and the preferences of the user.

To sum up, WIM tasks and capabilities are reusable because they are defined in a domain independent manner, thus the same agents can be used to build a new application with a new domain knowledge.

This chapter begins introducing the WIM approach to information search (§7.2) and briefly describing the WIM architecture (§7.3). Following, the main elements of the WIM application are described at the knowledge level: the information search ontology (§7.4) first, the specification of tasks and capabilities (§7.5) second, and the domain-models (§7.6) third. Besides, the chapter devotes an entire section to usage aspects such as the interfaces to the application (§**??**), another section to exemplify the use of the WIM library (§7.7), and another one describing an interlibrary application (§7.10) made in cooperation with other partners in the IBROW project.

7.2 The WIM approach to information search

The vast amount of information available in the Web causes some serious problems to the users when looking for information. It is difficult to find all the relevant information, for it is distributed among several information sources. In addition, the information retrieved is rarely ranked according to the users utility criteria. Meta-search (Figure 7.1) is one of the most promising approaches to solve the first problem. If the single search-engines store only a portion of all the existing information about some particular domain, several search engines should be queried to increase the search coverage. But often in practice, meta-search is under-exploited; existing meta-search engines do not combine results from the different single engines, neither they rank the information retrieved, or the ranking mechanisms are quite poor. Our approach overcomes these two limitations. The WIM application tasks allow to rank documents retrieved from information sources that originally do not perform any ranking; and it allows also to aggregate information retrieved from different search engines. The originality of our vision is that we exploit the filtering capabilities of existing search-engines, thus ranking is achieved with a little computation and storage cost.

These are, in brief, the main requirements we pose to the information search process:

- it should be possible to rank information retrieved from sources that are not able to rank information by themselves;
- to retrieve and combine information coming from multiple, heterogeneous, sources;
- to aggregate the rankings of information retrieved for several queries, so as to bring a unique, overall ranking; and
- to rank information according to the users utility, enabling for an easy modification or customization of the utility criteria.

WIM approach to information search can be described as an *adaptation* process with three main stages: query adaptation, information retrieval and aggregation. *Query adaptation* refers to the process of elaborating the user consultation to better fulfill his interest, as well as adapting queries expressed in terms of

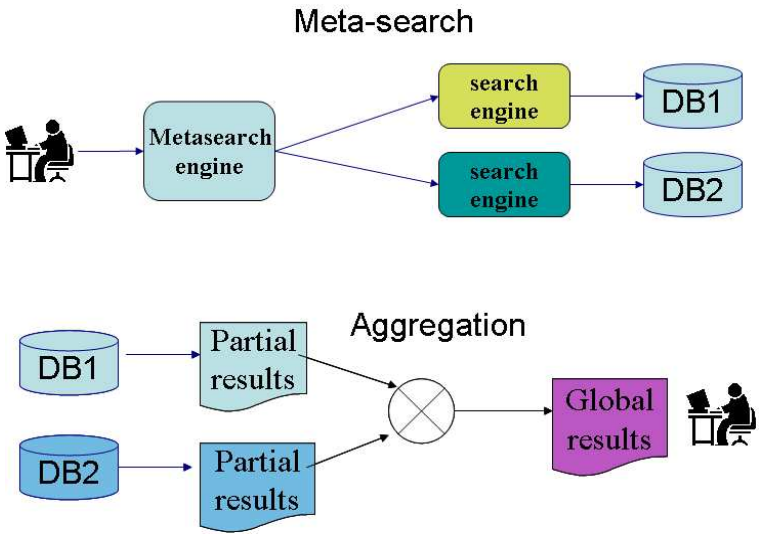


Figure 7.1: Meta-search and aggregation

a domain language (independent of any specific information source) to the language used by particular information sources. Once the retrieval is performed, the results from different queries are aggregated for each source, and finally the results for each source are aggregated again to obtain a unique result for the user consultation.

WIM approach to information search is based on *query weighting* and *numerical aggregation operators* [Gómez and Abasolo, 2003]. Query weighting refers to the process of assigning weights to queries generated according to some domain knowledge, while numerical aggregation operators are used to merge information retrieved from different information sources or different queries. Aggregation allows to combine the different ranking obtained by bits or items of information obtained for different queries, so as to obtain a unique ranking or score for each retrieved item. This approach allows to rank items retrieved from sources that originally do not give any ranking, and to define user-oriented utility measures simply by defining the appropriate knowledge categories (§7.6).

7.2.1 Adaptation of queries

WIM subscribes to the well known approach of representing queries as keyword vectors. This decision is justified because nowadays professional databases could often be accessed through the use of a web-based search-engine, whose queries are made of *keywords* belonging to a particular domain. In addition to keywords, WIM queries allow specifying search *filters* as optional constrains for narrowing the retrieval process. An ontology on bibliographic data has been used to model

the kind of filters allowed by professional bibliography search-engines, like *publication type*, *language* and so on.

There are two types of query: domain-query and source-query (§7.4.2). Domain queries are expressed in terms of the application domain, like medicine, while source queries are customized for a particular information source. In our implementation, the application domain is medicine, and the type of information retrieved is about bibliographical references. Therefore, WIM performs two types of adaptation:

- *Query elaboration* is performed at the domain level, using application domain knowledge to adapt queries to a particular user interest, within a particular domain, like medicine bibliography. Domain queries are elaborated using domain knowledge, like synonyms, hypernyms and hyponyms provided by a thesaurus, which can be used to obtain semantically equivalent queries, and to either generalize or specialize a query.

- *Query customization* is performed at the source level, using descriptions of information sources (Web-based search engines) to customize a query for a particular information source. Source queries are generated by translating keywords and filters from the domain level ontology into the *search modes* and filters supported by a particular information source.

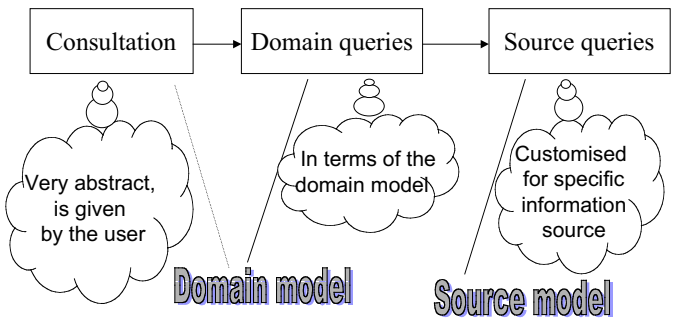


Figure 7.2: Domain and source query elaboration

Figure 7.2 sums up the idea of the query adaptation process at the two levels: domain level and source level.

1. The consultation introduced by the user is expressed as a domain-query, afterwards it is expanded into a collection of new domain-queries, using domain knowledge and utility criteria to elaborate queries. Resulting queries are weighted according to the similarity of the new domain queries to the user consultation.

2. Domain queries are transformed into source queries customized for specific information sources, using domain knowledge about the information

sources.

In our approach to query adaptation, new queries are not only enriched with domain knowledge, but they are also weighted according to the strength of the relationship between the original query and the new query (which depends on the type of query adaptation as well as on specific information represented as domain knowledge). For instance, query adaptation at the domain level (query elaboration) uses semantic similarities to generate new queries, while query adaptation at the source level uses information on the search modes supported by an information source and the type of filters applied. Once the query adaptation finishes, the selected information sources are queried through the use of wrappers, which are responsible for adapting the queries represented in terms of the WIM ontology to fit the particular interface of each information source. Queries generated during the query adaptation phase allow to score the items retrieved even though the information source does not give any ranking. Ranking can be straightforwardly applied because the queries are weighted in advance, during the adaptation process, thus their answers can be considered as inheriting the weight of the query as an assessment of relevance or utility.

A more formal description of the query weighting approach used by WIM information agents is provided as an Appendix in B.

7.2.2 Aggregation of results

The answers to all the queries are the items retrieved from different information sources, ranked according to the weights of the queries they are a result to. These items are combined to obtain a unique set of items, where repeated items are eliminated, keeping only one instance of each item with a unique, overall ranking.

Ranking synthesis is achieved by applying some aggregation operator. Aggregation is a kind of merging where the rankings assigned to different occurrences of an item are combined using an aggregation operator to obtain a unique ranking that summarizes the utility or relevance of each item for the user. A numerical aggregation operator is necessary because during the query adaptation process queries are weighted with numerical values. Hence, the results of a query (the retrieved items) inherit the weight of the query. If the queries are weighted according to some *utility* criteria rather than relevance, then the results are ranked taking into account these utility criteria. Notice that the same item may be retrieved as a result of several queries, and each query may have a different weight. Therefore, the aggregation process has to compute an overall score for each retrieved item based on the evidence degree contributed by the weights of the queries in which each item is retrieved. For this purpose, four numerical aggregation operators have been implemented as capabilities in the the library: the *arithmetic-mean*, the *weighted-mean*, the *Ordered Weighting Average (OWA)* and the *Weighted OWA* [Torra, 1996].

Sometimes, the term "fusion" is used to refer to the aggregation of items retrieved from different information sources. In our framework, the same pro-

cedure is used to aggregate items retrieved from a unique search-engine can be used in fusion. In this case, each source is assigned a weight expressing the reliability of that source or other kind of "goodness". A query customized for multiple search-engines is weighted using a combination of the weight assigned to the query during the query elaboration stage, and the weight assigned to the source in the source description.

7.3 WIM architecture

The WIM configurable application has been specified according to the ORCAS framework, both at the knowledge and the operational levels (i.e. using the ORCAS ACDL), and is configured through the institutional agents provided by the ORCAS e-Institution (Chapter 6).

WIM is based on a collection of domain-independent information agents registered in the ORCAS e-Institution as Problem Solving Agents (PSA), plus a repository of domain knowledge accessible by these agents.

We can distinguish three groups of agents in the WIM application:

- *Institutional agents* implementing the internal roles of the ORCAS institutional framework: Librarian, Knowledge-Broker and Team-Broker. These agents are common for any ORCAS-based application, they just offer the infrastructure to build a configurable application. We have built one agent for each role: there is one Librarian, one Knowledge-Broker and one Team-Broker.

- *Problem Solving Agents* equipped with capabilities to perform information search and aggregation tasks. These agents can participate in the institution by adopting the PSA role and registering their capabilities to a Librarian agent. From now on we will refer to the set of tasks and capabilities registered in WIM as the Information Search and Aggregation (ISA) library.

- *Personal Assistants* specialized in the WIM information tasks. These agents interact with the user (or a software agent) to obtain a specification of the problem, and bring back the results to the user (or agent).

In addition, there are several pseudo-agents providing connectivity to external agents and to non agent resources as well, namely: wrappers, FIPA-mediator and WWW-mediator.

- *Wrappers* are used to agentify external resources, like Web-based information sources (Pubmed, IGM-Healthstar, IGM-Medline, and iSOCO wrappers) and external domain knowledge (MeSH wrapper).

- The *FIPA-mediator* provides connectivity to external agents willing to access the WIM application.

- The WWW-mediator provides connectivity to an http protocol, and thus it supports Web based access to the WIM application.

However, the WIM library needs some domain models satisfying its knowledge requirements to become a complete application. We have specified as domain models some knowledge bases containing the knowledge on medicine and bibliographic data required by WIM capabilities to be applied. This knowledge is stored within a shared repository that can be accessed by any agent playing the PSA role.

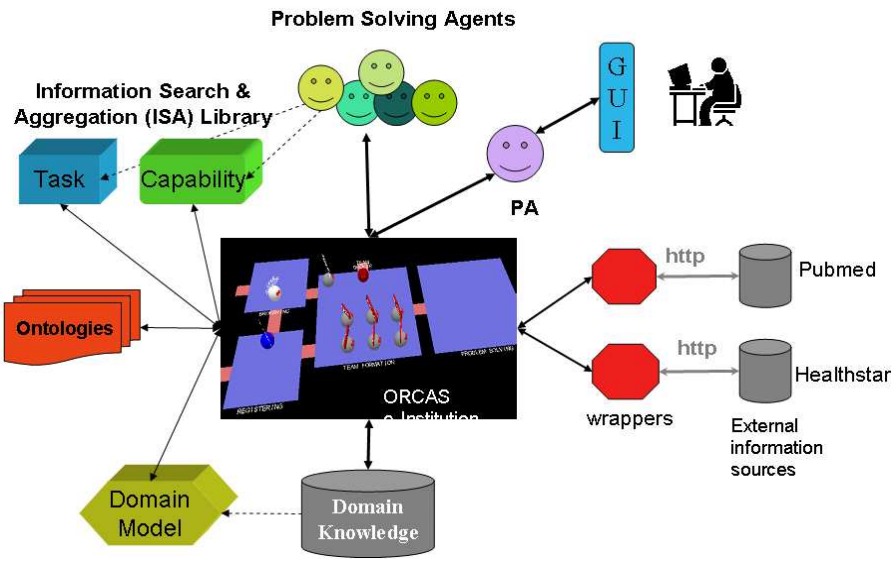


Figure 7.3: WIM architecture and interoperation

Figure 7.3 sums up the main elements of the WIM architecture integrated through the ORCAS e-Institution. There is a library on Information Search and Aggregation that is composed by the capabilities and tasks registered by several information agents (Problem Solving Agents), represented using some ontologies; there is a repository of domain knowledge characterized by domain-model; and there are some wrappers allowing agents to interoperate with external information sources. Moreover, there is a Personal Assistant agent responsible for processing mediating between the user and the application. The ORCAS e-Institution is the central element, constituting a middleware layer where agents meet to interact according to the shared social knowledge (ontologies, communication language and interaction protocols) provided by the institution. The ORCAS e-Institution provides an added value to both providers and requesters through a collection of middle agents that mediate between requesters and providers: Librarian, Knowledge-Broker, Team Broker, and Personal Assistant.

WIM interface to external agents is based on the idea of services. Each service defines a type of operation that can be requested by other agents, and is specified by an interaction protocol and a data format specified according to the FIPA proposals. WIM services specialize the ORCAS services to handle concepts from the the Information Search and Aggregation (ISA) ontology, as described in Appendix F.

7.4 The Information Search and Aggregation Ontology

Ontologies are extensively used through WIM including the following:

- the Knowledge-Modelling Ontology (KMO) at the knowledge level;
- the ontologies defined by the ORCAS e-Institution, which are the Teamwork Ontology and the Brokering Ontology; and
- the ontology used to specify the components in the ISA-Library (the ISA-Ontology) and the ontologies used to specify the domain-models used in WIM.

The concepts of the Knowledge-Modelling Ontology are described in Chapter 4 and Appendix A. The ontologies used within the ORCAS e-Institution are described through several sections of Chapter 6. This section describes the ontologies used to specify the ISA-Library, that is to say, the vocabulary used to specify the features characterizing the tasks and capabilities in the WIM application. Recall that this vocabulary is represented using the Object Language, and we are using of Feature Terms (§4.3).

The ontologies and the components expressed in terms of those ontologies are both represented using the NOOS [Arcos, 1997] representation language. There are several reasons endorsing this decision. First, the ORCAS KMF provides and architectural framework that is neutral about the content language (the Object Language). Second, there is an implementation of an agent platform in NOOS that is compatible with FIPA, so describing the ISA-Library in NOOS facilitates the development of an agent based system for the WIM application. Morevoer, the NOOS agent platform has some facilities to translate XML-based information and is an object-centered representation language; therefore NOOS is compatible with RDF-based representations like those used in describing Semantic Web Services (e.g. DAML-OIL) and by the Protégé editor suite used in Knowledge Modelling and Ontology Engineering.

The approach taken here has been to depart as few as possible from the core assumptions of the ORCAS KMF while using NOOS in a easy and understandable way. For this purpose, we have designed a few macros on top of the NOOS language; these macros are used to generate the NOOS code in the agent-based implementation of the ISA-Library. The point here is to make agents use ORCAS concepts as part of their content language in the agent communication language.

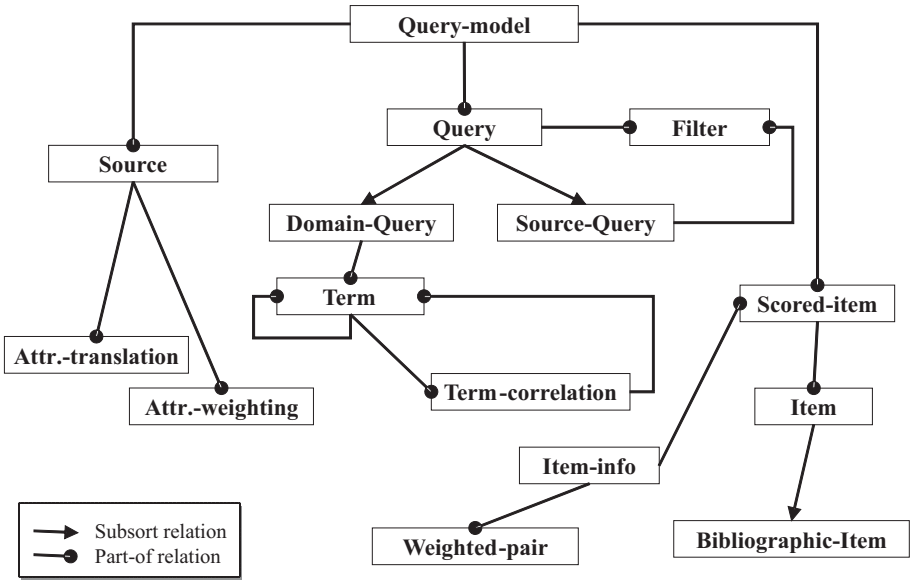


Figure 7.4: Overview of the ISA Ontology

The ISA-Ontology consists of two different parts: one encompasses the concepts used as signature elements and is used to describe the inputs, outputs and knowledge-roles, and the other encompasses the formulae and is used to specify the preconditions, postconditions and assumptions.

There are two sorts that are not subsorts of other sorts: the sort FT-Signature-Element, which is used to specify signature elements in the Object Language of Feature Terms, and the sort FT-Formula, which is used to specify the formulae (realize that these sorts are refinements of the abstract sorts defined by the Knowledge Modelling Ontology, named Signature-Element and Formula).

The main sorts of the the ISA-Ontology used to specify signature elements in the WIM library, and the relations among these sorts are drawn in Figure 7.4. These sorts are described step-by-step in the following subsections, while the formal specification is included in Appendix D. The concepts used to specify formulae are not described as a separate entity; instead, some of them are introduced as needed when describing the tasks and capabilities in the ISA-Library (§7.5).

7.4.1 Items

An *item* (Figure §7.5) is the informational unit used during the information search tasks. It is defined in very generic terms, so as to be refined for different purposes, as follows:

The sort item has the following features:

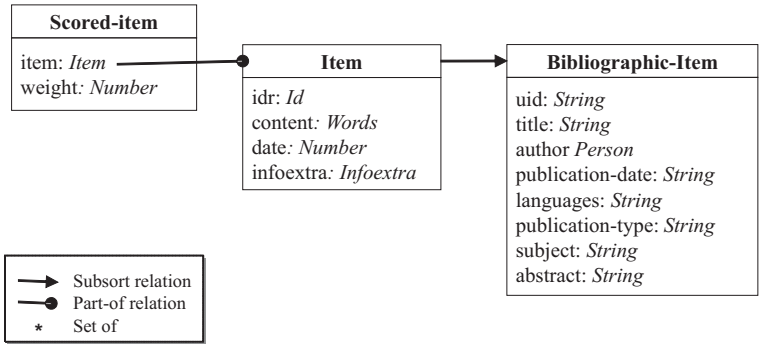


Figure 7.5: Sort definitions of Item, Scored-item and Bibliographic-item

- *Identifier (Id):* it is used as a reference to the piece of information embodied by the item. An item exists initially with a value in its *Id* slot and the remaining slots empty. As long as the search processes take place, the rest of fields are gradually fulfilled.

- *Content:* it corresponds to the actual piece of textual information (e.g. lists of words) that will be gathered from external sources. Its format depends on the source it comes from. For example, a typical content in Internet is an HTML page (and following this Internet analogy, the *Id* would correspond to the URL) whereas in a Data Base, a content will correspond to a specific record and the ID would be the index number.

- *Date:* it is necessary to keep track of the moment that an item of information is obtained or updated. Their values are numerical (they represent number of seconds) so that numerical operators can be easily applied to compare two dates.

- *InfoExtra:* There is additional information used to check if a content should be updated yet or not. For example, this field includes a code for any problem obtaining the content and the language of the content.

The sort Item can be refined to deal with specific types of information; specifically, we are interested in searching bibliographic information, so we have defined a Bibliographic-Item sort that introduces bibliographic data within the specification of an item. Figure §7.5 shows the sort definition for the sorts Item and Bibliographic-item, which is defined as a subsort of the former. The sort Bibliographic-item adds some slots to those inherited by the Item sort, namely a unique identifier (UID), a title, a set of authors, a publication date, a set of languages, a publication type, a subject (usually specified as a set of *keywords*), and an abstract.

A *scored-item* is a pair that associates an item with a number which represents its *score* with respect to the user consultation. The score of an item may

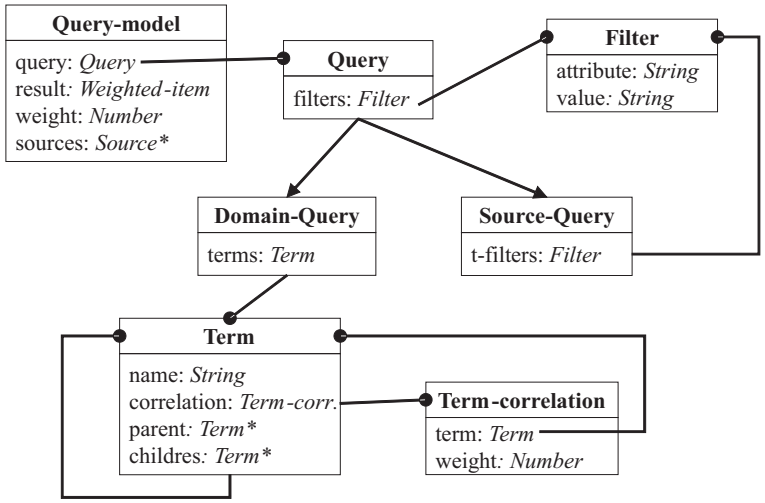


Figure 7.6: Sort definitions of Query, Filter, Term, and Query-model

be given as the result of a query to a source or as the result of an aggregation process over the results of several queries.

7.4.2 Queries, Filters and Terms

The information about what is being searched on behalf of a user is expressed in this ontology by queries, filters, terms and categories. Figure 7.6 shows these sorts and their relationships.

Queries

The general sort query represents the user's consult. There are several types of queries that share only a feature, called *filters*.

The user's consults are specified as instances of the sort Domain-query that in addition to filters contains a list of terms and one category.

When the domain query is translated to the concrete vocabulary of the source to be accessed, i.e. it is *customized*, the query is of sort source-query. The information in a domain-query is translated to *filters* and *t-filters* in a source-query.

Filters

Filters are constraints over objects of the search. They are represented by the sort filter that has two features:

- an *attribute*, which specifies the name of a filter to constrain the search of information; and

- a *value* for that attribute, to be satisfied by the the results to retrieved.

For instance, let us suppose that our goal is to search in the medical database Medline for documents published after 1990. A filter of a domain query will contain as *attribute* "begin year" and as *value* "1990".

Terms

Terms are words pertaining to an ontology used in a query. There are relations between terms (in a ontology) that embody domain knowledge and, as such, they are located in a domain model that uses this ontology. These relations between terms are included in the features *parent* and *children* of sort Term (Figure 7.6).

The feature *parent* represents the relation of a term with other more general terms. The *children* feature represents the relation of a term with more specific terms. The feature *term-correlation* represents the relation between two terms. The sort Term-correlation has two features : *term* and *weight*. *Weight* is the correlation degree of *term* with the current term. When the weight is 1 it means that both terms are synonyms.

Categories

A category is an intensional definition of a class of search objects in the context of a particular domain ontology. For instance, we have specified some categories concerning clinical medicine such as *diagnosis*, *therapy*, and *clinical guidelines* (Figure 7.12).

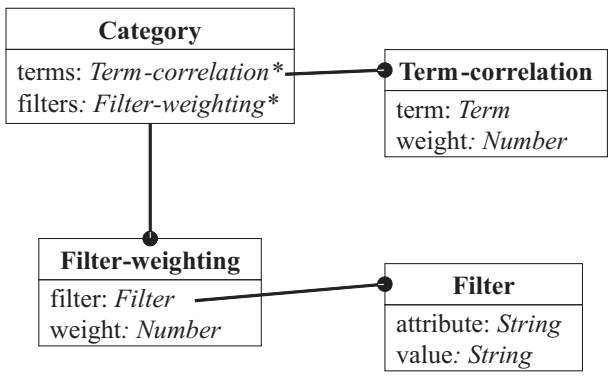


Figure 7.7: Sort definition of Category

A *category* (Figure 7.7) express its intensional meaning as a collection of correlated *terms*, which are specified as elements of sort Term-correlation, and a collection of filters, which are specified as elements of sort Filter-Weighting. A Filter-Weighting is a pair composed of a *filter* and a *weight* .

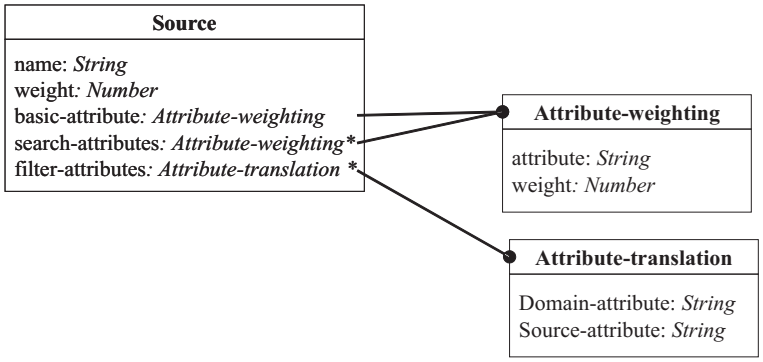


Figure 7.8: Sort definitions of Source, Attribute-weighting, and Attribute-translation

7.4.3 Sources

We consider a source as composed of two parts: the content of a database (for instance Medline or HealthStar) and a particular search engine that can be used to access that content (for instance PubMed or IGM). A source is modelled using the sort source that has the following features:

- *Name*: that represents the name of a source.
- *Weight*: a weight assessing the reliability of a source.
- *Search-attributes*: the set of attributes allowed within a query the source. These attributes may be record fields of a database or search modes of the search-engine. Search modes indicate accesses that are combination of several fields. For instance, the attribute "subject" of the IGM method searches a word both as a MesH term and in the abstract.
- *Basic-attribute*: represents the basic or default search mode of a source For instance, the basic attribute of the PubMed-Medline is called "all", which means that the search is performed using all the available search fields, thus it is the more general way of searching Medline.
- *Filter-attributes*: is a set of attributes that can be applied in order to limit the search. Some examples of filter attributes are the kind of publication, date, language, etc.
- *Content*: The name of the database. For instance, Medline that is a database on medical bibliographical data.

The features *name*, *search-attributes*, *basic-attribute*, and *filter-attributes*, are referred to the method that allows to access the database.

The values in *search-attributes* are of sort Attribute-weighting, which has two features: *attribute* and *weight*. The *weight* indicates the semantic importance of

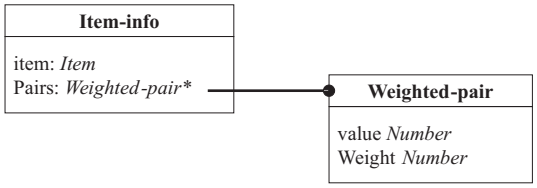


Figure 7.9: Sort definition of Item-info

the *attribute*. For instance, in the Medline information source the weight of the feature *title* is greater than the weight of the feature *abstract*. This means that the same keyword is more important when it appears in the title than when it appears in the abstract.

The feature *filter-attributes* is specified by elements of type Attribute-translation (see Fig. 7.8), which is composed of a *domain-attribute* and a *source-attribute* (both of sort String), and specifies a mapping schema between the domain ontology and the source ontology.

7.4.4 Query-models

The *Query-model* sort (Figure 7.6) encapsulates a query and all extra information to be associated to a query during the performance of the task Information Search, which includes the following features:

- *Query:* the query we are talking about (§7.4.2)
- *Source:* the source to which this query is addressed (§7.4.3)
- *Result:* a set of *scored-items* (§7.4.1) returned by the *source* as answer to *query*.
- *Weight:* the weight assessing the utility or relevance of the *query*
- *Children:* the set of query models related with the *query*. For instance, the query model representing the user's consult is related with query models containing elaborated queries.

7.4.5 Item-info

The aggregation capabilities need all the information about an item in order to calculate a global score for that item. This information is represented by the sort Item-info, which is composed of an *item* and a collection of *pairs*. The feature *pairs* (see Figure 7.9) contains pairs *value-weight* where *value* is the importance of the item and *weight* is the importance of the query that has produced the retrieval of the item.

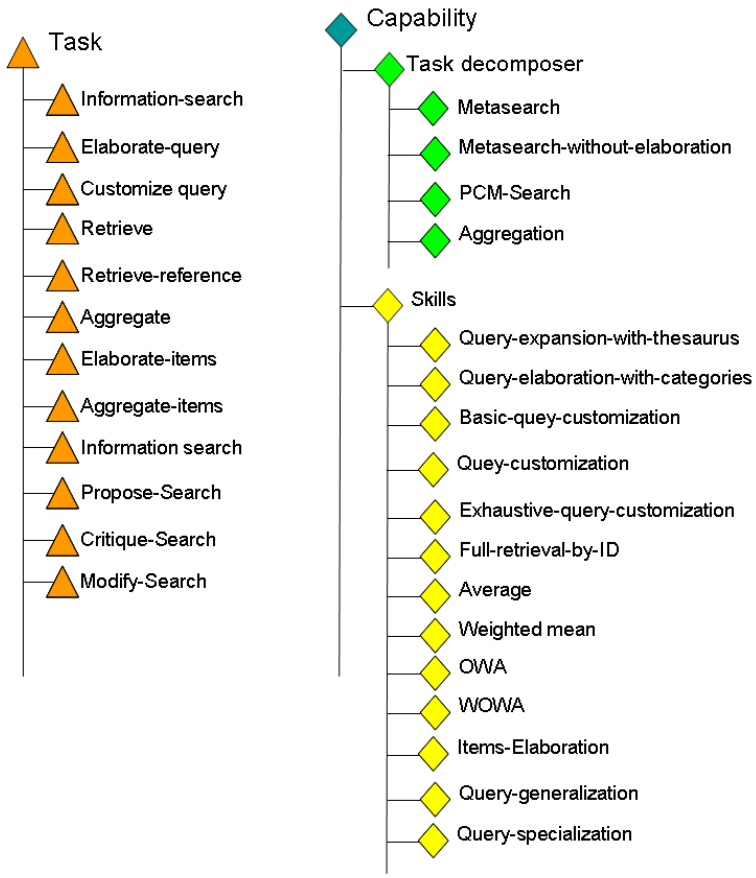


Figure 7.10: Hierarchy of components in the WIM library

7.5 The WIM library

The WIM library is composed of a collection of tasks and capabilities concerning information search and aggregation of information in the Internet. Figure 7.10 shows the hierarchy of components in the WIM library, including 14 tasks, 10 skills, and 5 tasks decomposers. These components are all instances of a sort in the Knowledge-Modelling Ontology, whether it is the Task, Skill, or Task-Decomposer sort.

Tasks are decomposed into subtasks by tasks-decomposers following a hierarchical task/subtask decomposition schema. The top-level task is called Information-search, and have the goal of obtaining a set of items answering a user's consult. Commonly, the user's consult can be expressed as a set of keywords and filters, and several kinds of results are possible (e.g. documents, Web

pages, images, etc.). Within this section the different tasks and capabilities of the ISA-Library are described in terms of the ontology described in the previous section.

Before to go through the specification of components in the library some notes on the Object Language are pertinent. We have considered two formalisms as the Object Language: Lisp-like predicates (*predicate* $?var_1$ ... $?var_N$) and Feature Terms (see §4.3). Although we have used only the Features Terms formalism in the final implementation of WIM due to pragmatic consideration, some examples of the predicate-based representation will be provided at some points for illustration purposes.

7.5.1 Information Search task

```
(define (Task :id Information-Search)
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Domain-Queries))
   (define (var)
     (name 'available-sources)
     (sort Source)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-CONSULT
      ))))
```

The input for the task Information Search is a consultation by the user, specified as a Domain-Query and a collection of information sources, specified by elements of sort Source. The output is a collection of elements of sort Scored-Item and the goal is to obtain items satisfying the user consultation, which is represented by the formula SATISFY-CONSULT, which is a subsort instance of the sort Formula-FT (from now onwards formulae identifiers are specified using capitalized characters).

There are several capabilities that are suitable for the Information-search task, where suitable means "being able to solve", as established by a *task-capability matching* relation (Definition 4.1). Specifically, there are 4 capabilities suitable for the task Information-Search, and all of them are task-decomposers:

1. Metasearch,
2. Propose-Critique-Modify-Search (PCM-Search),
3. Metasearch-with-source-selection, and
4. Metasearch-without-elaboration.

We are going to describe here the Metasearch and the PCM-Search capabilities. The Metasearch capability is a task-decomposer that decomposes the task Information-Search in four subtasks: Elaborate-query, Customize-query, Retrieve and Aggregate.

```
(define (Task-Decomposer :id Metasearch)
  (name "Metasearch")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Domain-Queries))
   (define (var)
     (name 'available-sources)
     (sort Source)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-CONSULT
      )))
  (subtasks
   Elaborate-query
   Customise-query
   Retrieve
   Aggregate))
```

Recall that a task-capability matching is defined at the knowledge level as a combination of signature and specification match (Definition 4.1):

$$match(T, C) = (T_{in} \geq C_{in}) \ \wedge \ (T_{out} \leq C_{out}) \ \wedge \ (T_{pre} \Rightarrow C_{pre}) \wedge (C_{post} \Rightarrow T_{post})$$

The former relation is specialized for the Feature Terms Object Language using subsumption as the inference mechanism (Definition 4.4):

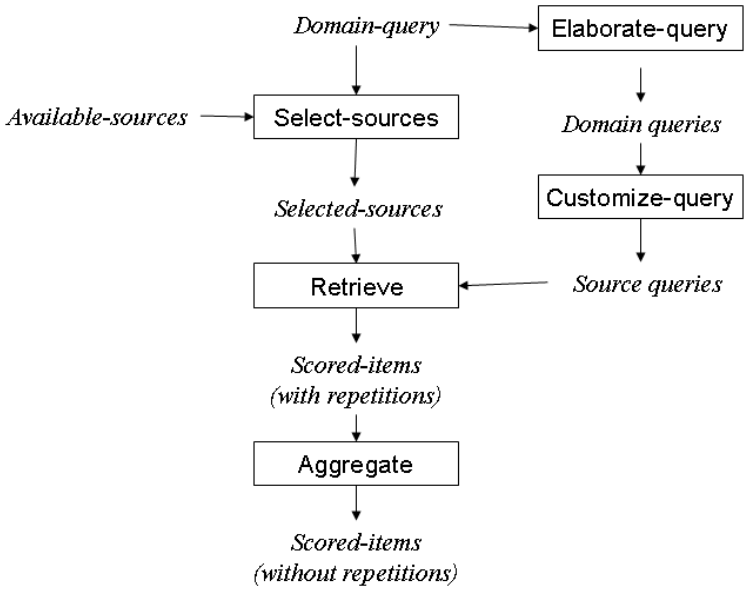


Figure 7.11: Overview of the capability Metasearch-with-source-selection

$$match(T, C) = T_{in} \sqsubseteq C_{in} \wedge C_{out} \sqsubseteq T_{out} \ \wedge C_{pre} \sqsubseteq T_{pre} \wedge T_{post} \sqsubseteq C_{post}$$

The former relation holds between the Information-Search task and the Metasearch capability; specifically, there is an exact match rather than a plug-in match between them, that is to say: the input, output, preconditions and postconditions of the tasks are exactly the same than those of the capability.

The capability Metasearch-without-elaboration is similar to Metasearch, but omits the task Elaborate-query; moreover, the Metasearch-with-source-selection capability is also similar to Metasearch but introduces a new task dealing with the selection of a subset of information sources from the set of available sources.

However, the capability PCM-Search introduces three subtasks: Propose-Search, Critique-Search and Modify-Search. The first and the third of the former tasks are similar to the task Information-Search, thus they can be solved by any of the two Metasearch capabilities.

Figure 7.11 shows the subtasks introduced by the task-decomposer Metasearch-with-source-selection and the data flow among subtasks, which is represented by the sorts of their inputs and outputs.

The task Elaborate-query takes as input the user's consult (a domain-query embedded within a query-model) with the goal of generating new queries so as to increase the recall and the precision of the original query. The idea of this task is to enrich the user's consult using semantic relationships between terms in the query and terms within a knowledge repository such as a thesaurus.

The queries resulting of the Elaborate-query task may be asked to all the available sources or only to a subset of them. For this reason there is a subtask called Select-sources, which takes a collection of sources (available sources) as input and has the goal of determining the subset of those sources where it's worthwhile to search information for a particular consult (selected sources).

The Customize-query task embodies the process of adapting the domain-queries obtained by the Elaborate-query task to the idiosyncratic features of a specific information source. As a result, a set of source-queries are generated as output and further used as input of the *Retrieve* task together with the set of selected-sources resulting of the Select-sources task.

The Retrieve task is performed on query-by-query basis, for it takes a single source-query and a source as inputs. The result of the Retrieve task is a set of scored items retrieved from that source (embodied within the *query-model* containing the *source-query*). Since that task may be performed several times, several collections of items would be retrieved, probably involving repeated apparitions on the same item with a different weight. The next task deals with this multiplicity of gathered information.

The Aggregate task has the goal of integrating all the information about an item retrieved as a result of different queries, perhaps obtained from different sources. The input of the Aggregate task encompasses all the scored-items contained within the query-models resulting of the many performances of the Retrieve task. Each *query-model* has a set of *scored-items* in the *results* slot, i.e. the items retrieved from one source and their associated weight. The goal of the Aggregate task is to eliminate repetitions and to obtain an overall weight for ranking each of the retrieved items —detecting whether two items refer to the same information and using the capabilities on aggregation (see §7.5.8). Thus, the output of this task —that is also the output of the information-search task— is a single set of scored-items.

In the following sections the subtasks of the task-decomposer Metasearch-with-source-selection are described, as well as some related capabilities (some capabilities suitable for the those subtasks).

7.5.2 Elaborate-query task

```
(define (Task :id Elaborate-query)
  (name "Elaborate-query")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name '?consult)
     (sort Domain-Query)))
  (output-roles
   (define (var)
     (name '?elab-queries)
     (sort Domain-Query)))
```

```
(competence
 (define (Competence)
   (postconditions
    ELABORATE-CONSULT))))
```

The Elaborate-query task takes the user's consultation (*?consult*) as input, specified as a Domain-query, and has the goal of generating new domain queries (*?elab-queries*) as output, so as to increase the recall and the precision of the original query. The idea of this task is to enrich the user's consult using semantic relationships between terms in the query and terms within a knowledge repository such as a thesaurus. The elaboration of a query is useful when the user wants to search documents about a topic that can be expressed using different, alternative keywords and filters. In order to go through the Elaborate-query task, a capability generates new queries taking into account some domain knowledge (e.g. a *thesaurus*) without the user having to declare it. The output of the Elaborate-query task is a set of elements of sort Domain-query.

An alternative specification of the Elaborate-query task is provided below using a predicate-based representation in place of the Feature Terms formalism.

```
(define (Task :id Elaborate-query
  (ontology ISA-Ontology)
  (input (?consult :sort Query-model))
  (output (?elab-queries :sort Query-model))
  (preconditions (subsumes domain-query ?consult.query))
  (postcondition (forall (?qm in ?elab-queries)
                   (elaborated-from ?qm.query ?consult.query))))
```

There are four capabilities, all of them skills, suitable for the Elaborate-query task:

1. Query-expansion-with-thesaurus,
2. Query-expansion-with-categories,
3. Exhaustive-query-expansion-with-thesaurus,
4. Exhaustive-query-expansion-with-categories,

The capability Query-expansion-with-thesaurus is a skill that generates new queries in which some terms —the keywords— are replaced by synonyms or related terms, and the queries are weighted according to the semantic similarity between the original term and the new term (represented with a correlation coefficient). The capability Query-expansion-with-categories does not replace terms by other terms; this method generates new queries by adding new keywords and filters to the user consultation according to a category (a topic) selected by the

user. The knowledge used to enrich the queries is obtained from a collection of predefined categories or topics. The exhaustive versions of the former capabilities do the same work but they generate a greater number of queries by using more complex variations of the original query (replacing more than one keyword or filter per query).

```
(define (Skill :id Query-expansion-with-thesaurus)
  (ontologies ISA-Ontology)
  (input-roles
  (input-roles
   (define (var)
     (name '?consult)
     (sort Domain-Query)))
  (output-roles
   (define (var)
     (name '?elab-queries)
     (sort Domain-Query)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-WITH-THESAURUS)))
  (knowledge-roles
   Thesaurus))
```

The Query-expansion-with-thesaurus skill takes a domain-query as input (*?consult*) and produces a collection of domain-queries as output (*?elab-queries*); the goal is to build new goals by changing or adding elements of the original query, which is represented by the formula ELABORATE-WITH-THESAURUS. This formula means that the new queries are modifications of the input query using knowledge from a thesaurus (synonyms and related words), so a knowledge-role of sort Thesaurus is specified within the *knowledge-roles* slot. If we compare the specification of this skill with the specification of the Elaborate-query task we realize that they match, though it is not an exact match but a plug-in match.

A task-capability matching holds between the Information-search task and the Query-expansion-with-thesaurus skill, though there is not an exact competence match, but a plug-in match. Specifically, the formula ELABORATE-CONSULT specified as a postcondition of the task and the formula ELABORATE-WITH-THESAURUS satisfy the subsumption relation established by a task-capability matching, for the sort of ELABORATE-WITH-THESAURUS is a subsort of the sort of ELABORATE-CONSULT (Figure 4.14), and thus the condition $T_{pos} \sqsubseteq C_{pos}$ holds.

An alternative version of the former skill is provided below using a predicate-based representation.

```
(define (Skill :id Query-expansion-with-thesaurus
  (ontology IS-Ontology)
  (input (?consult :sort Query-model))
  (output (?elab-queries :sort Query-model))
  (knowledge thesaurus)
  (preconditions (subsumes domain-query ?consult.query))
  (postconditions
    (forall (?el-q in ?elab-queries)
      (and (elaborated-from ?el-q.query ?consult.query))
           (uses-synonyms ?el-q.query ?consult.query))))
```

Again, one can verify that a task-capability matching relation holds between the Information-search task and the Query-expansion-with-thesaurus skill, though a renaming of *?el-q* to *?qm* is required. If we compare the postconditions of both specifications we found that the postconditions of the capability imply the postconditions of the task, since they are more specific (notice the capability incorporates an additional clause in the postconditions with the predicate *(uses-synonyms ?el-q.query ?consult.query)*.

7.5.3 Select-sources task

Given a domain query and a set of available sources, the goal of the Select-sources task is to determine a subset of the input sources useful for the query.

There are several capabilities for solving the Select-sources task. The simplest one is Ask-user; this skill shows the available sources to the user and lets him choosing the subset to be used for retrieving information. This capability is used when the user has some knowledge about which sources may be more useful.

Another capability for solving the Select-sources task is the Case-based-source-selection capability, which is based on using similarity measures. The capability Case-based-source-selection imports a domain model that has a case base and a similarity measure. Each case represents a domain query that has been successfully asked to a specific source.

7.5.4 Customize-query task

The Customize-query task deals with the adaptation of domain queries to the features of a specific information source. The goal of the customize-query task is to translate a query expressed using the terms of the domain into a collection of new queries expressed in the terminology of a particular source. The customization focuses on the attributes of the filters used in the input query. There are two possible cases: 1) an attribute is allowed by the source, and 2) an attribute has to be translated to the ontology used by the source. In case 1) no translation is needed, and in case 2) the capability uses knowledge storing a description of the source to know which are the mapping schemes between the domain vocabulary and the source vocabulary. One of the attributes of a

source is the collection of search modes allowed. The point is that the keywords in the domain query may be searched using different modes or strategies, hence more than one source query can be generated for the same domain query using different search strategies and trying alternative filters. The purpose of trying different search strategies and filters is to obtain more information to rank the items retrieved. For instance, if search information for a query using a very general search mode and then repeats the same query using a more restrictive search mode, one can compare the items appearing in both queries to determine which results are more relevant to the query (the ones appearing as a result of the two queries, see §7.7).

```
(define (Task :id Customize-query)
  (ontologies ISA-Ontology)
  (input
   (define (var)
     (name '?query)
     (sort Domain-Query)
   (define (var)
     (name '?source)
     (sort Source)))
  (output-roles
   (define (var)
     (name '?s-queries)
     (sort Source-Query)))
  (competence
   (define (Competence)
     (postconditions
       CUSTOMIZE-DOMAIN-QUERY))))
```

The input of this task consists of a query of sort Domain-query (as those resulting from the Elaborate-Query task) and one specific source from those ones specified in the user's consultation or selected by the Select-source task. The output is a collection of source queries (*?s-queries*) which are customized for the input source, as expressed by the formula CUSTOMIZE-DOMAIN-QUERY.

There are three capabilities, all of them skills, that match the Customize-query task:

1. Query-customization,
2. Exhaustive-query-customization, and
3. Basic-query-customization.

```
(define (Skill :id Query-customization)
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name '?query)
     (sort Domain-Query)
   (define (var)
     (name '?source)
     (sort Source)))
  (output-roles
   (define (var)
     (name '?s-queries)
     (sort Source-Query)))
  (competence
   (define (Competence)
     (postconditions
      NON-EXHAUSTIVE-CUSTOMISATION
      )))
  (knowledge-roles
   Source-Descriptions))
```

7.5.5 Retrieve task

The goal of the Retrieve task is to effectively retrieve from a source a set of items satisfying the input query. The Retrieve task takes a source-query and a source as input, and produces a set of scored-items as output (specified within a Query-Model, in the *results* slot), such that an item's score expresses the degree the item satisfies the input query.

```
(define (Task :id Retrieve)
  (name "Retrieve")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Source-Query)
   (define (var)
     (name 'source)
     (sort Source)))
  (output-roles
   (define (var)
     (name 'result)
     (sort Query-Model)))
  (competence
```

```
(define (Competence)
  (postconditions
   SATISFY-QUERY))))
```

There is only one capability suitable for the Retrieve task, the Wrapper-based-retrieval skill, with a specification matching exactly the specification of the Retrieve task. We have decided to use Internet as the place to look for information; specifically, we focus on Web-based search engines dealing with medical bibliographic databases, such as *Medline* (accessed through the *Pubmed* search engine) and *Healthstar* (accessed through the *Internet Grateful Med* search engine).

From this point of view, the real retrieval capabilities are the Web-based retrieval engines, and the Wrapper-based-retrieval capability is a mere bridge to a source-specific wrapper that agentifies the Web-based search engine by handling the query and parsing the html results to obtain a structured representation of the information.

7.5.6 Aggregate task

```
(define (Task :id Aggregate)
  (name "Aggregate")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-models)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-ALL))))
```

The goal of the Aggregate task is to achieve a coherent and unique set of scored items from all the items retrieved from all the queries originated from the user's consult. The input for this task is a collection of query-models, each one containing the results concerning a single source-query. The output is a set of scored items, where all the information concerning the same piece of information is aggregated to obtain a unique scored item summing up the scores assigned to the different references to the same information unit.

There is a capability suitable for this task, the Aggregation task-decomposer. This task decomposer introduces two subtasks: Elaborate-item-infos and Aggregate-item-infos.

The Elaborate-item-infos task (see §7.5.7) transforms the information contained in a collection of query-models into a collection of item-infos, each one gathering all the information about a single piece of information originally distributed among several query models. The purpose of this task is to represent the information in a format more appropriate for the numerical aggregation operators implemented in the ISA-Library (see §7.5.8). The Aggregate-item-infos subtasks receives (iteratively) the information about an item (an item-info) and uses a mathematical function like a mean to compute a single score for that item. For this reason, the capability Aggregation performs an iteration of the Aggregate-item-infos subtask over the set of item-infos that are the output of the Elaborate-item-infos subtask.

```
(define (Task-Decomposer :id Aggregation)
  (name "Aggregation")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-models)
     (sort Query-model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-ALL
      )))
  (subtasks
     Elaborate-items
     Aggregate-items))
```

Notice there is an exact match between the Aggregate task and the Aggregation capability.

7.5.7 Elaborate-item-infos task

```
(define (Task :id Elaborate-items)
  (name "Elaborate-items")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-models)
```

```
    (sort Query-Models)))
  (output-roles
   (define (var)
     (name 'item-infos)
     (sort Item-Infos)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-ITEM-INFOS))))
```

The Elaborate-item-infos task has the goal of grouping all the information about the same unit of information (e.g. the same bibliographical reference) that is distributed among several query models as a result of retrieval information several times. The objects encompassing all the information on a single unit of information are created as instances of the sort Item-info. The input for this task is a set of query models containing a set of scored items each one, and the output is a set of item-infos.

There is only a capability suitable for this task, the Items-elaboration skill. This capability operates by taking the results (the scored items) of a collection of queries (represented as query models) and looking for those items referring to the same piece of information, like a bibliographic reference. As a result of the query expansion process carried out for the Elaborate-query task and the Customize-query task, the same piece of information may have been retrieved for different queries, thus appearing as different items. The goal of this capability is to detect whether different scored items refer to the same piece of information in order to eliminate redundancy. However, the aggregation of the different scores assigned to the same piece of information is performed by a capability suitable for the Aggregate-item-infos task. The Items-elaboration skill keeps a set of item-infos, one for each new piece of information, and iteratively compares each scored item with the set of item-infos to decide whether they refer to the same piece of information; if the answer is yes, the information on the item is added to the item-info, else a new item-info is created. Item infos encompass two types of information: weights and scores. Weights are obtained from the weights assigned to the query models where an item appears, whilst scores are assigned during the retrieval process by the information source or assigned on a default basis: a maximum score of 1 for each apparition of an item within a query-model, and 0 for each absence. In conclusion, an item info is a collection of pairs composed of a value and a weight (v_i, w_i), the former representing the relevance assigned to an item by the retrieval engine, and the later expressing the combined utility and relevance estimated during the elaboration and the customization of a query.

7.5.8 Aggregate-item-infos task

The Aggregate-item-infos task is a purely mathematical task and the capabilities that can solve this task are capabilities embodying mathematical aggregation functions like a *weighted mean*. The input of this task is a collection of weighted-pairs (v_i, w_i) consisting of a *value* (v_i) and a *weight* (w_i). The goal of this task is to obtain a single value for all the items referring to the same piece of information by aggregating its weighted values. Within the context of the Aggregation task-decomposer, this task is applied several times, one for each Item-info obtained during the Elaborate-item-infos task. Recall that an item-info encompasses all the information retrieved about a piece of information, like a bibliographical reference.

There are several methods that can be found in the literature for aggregating information based on numerical aggregation operators functions. We have included four methods in the library to perform this task: the *average*, the *weighted mean*, the *ordered weighted averaging* (OWA) and the *weighted ordered weighted averaging* (WOWA).

The *average* applies an arithmetic mean. The *weighted mean* is a lineal combination of values according to a vector of normalized weights. The weighted mean expects a number (say n) of weighted pairs to obtain a unique, aggregated value:

$$\frac{\sum_{i=1}^{n} v_i w_i}{\sum_{i=1}^{n} w_i}$$

The *ordered weighted averaging* (OWA) operator is, as the weighted mean, a lineal combination of the values according to a vector of weights. However, the difference is that OWA has a *weighting function* that establishes weights as a function of an ordering of the values to be aggregated.

First, using the OWA operator, the input values $\{v_i\}$ are (increasingly or decreasingly) ordered, giving an ordered collection of values $\{v_{\sigma(i)}\}$ where σ is the permutation performed. Then a *weighting function* Q assigns a weight to a value according to its position in that order:

$$\omega(i) = Q\left(\frac{i}{n}\right) - Q\left(\frac{i-1}{n}\right)$$

Finally, a lineal combination of the values with the assigned weights is computed:

$$\sum_{i=1}^{n} \omega_i v_{\sigma(i)}$$

OWA is parametric with respect to the weighting function: domain knowledge establishes the weighting function to be used in an application domain. Our approach is that the capability OWA imports a domain model or a particular application where the weighting function to be used is given as part of the domain knowledge.

The *weighted ordered weighted averaging* (WOWA) operator is a generalization of both the *weighted mean* and the OWA.

As in the OWA operator, the values $\{v_i\}$ are (increasingly or decreasingly) ordered. Then they are weighted according to their order using a weighting function, giving an ordered collection of values $\{v_{\sigma(i)}\}$ where σ is the permutation performed.

However, in addition to the weights derived from the order, WOWA uses the weights associated to the values in the weighted pairs. The combined weight ω is computed as follows:

$$\omega(i) = Q\left(\sum_{j\leq i} w_{\sigma(i)}\right) - Q\left(\sum_{j<i} w_{\sigma(i)}\right)$$

where $w_{\sigma(i)}$ denotes the permutation performed by the ordering upon the weights of the pairs (v_i, w_i).

Finally, a lineal combination of the values with the combined weights ω_i is computed as follows:

$$\sum_{i=1}^{n} \omega_i v_{\sigma(i)}$$

Let us remark here that the selection about which aggregation capability should be used is based on the preferences of the user as well as the type of information retrieved.

7.6 WIM domain knowledge

WIM uses domain knowledge from the fields of medicine and bibliographic data. This domain knowledge has been characterized by a collection of *domain-models*, which are specified by an ontology, plus a collection of properties and meta-knowledge, as described in §4.2.1 (Figure 4.12).

The main task for the WIM application is searching medical literature, and the utility criteria used to rank documents can be provided either by a medical thesaurus or by a collection of specific knowledge categories. In addition, in order to specify queries from an abstract, conceptual view, WIM needs bibliographic knowledge so as to specify queries to bibliographical search engines and handle bibliographical information. These are the three types of domain knowledge included in the WIM application, and thus there are three domain-models in WIM, one for each type of knowledge, namely the *MeSH* thesaurus, a categorization of *Evidence Based Medicine*, and a description of several bibliographical databases accessible through the Internet.

- MeSH is a general *medical thesaurus* that contains a huge collection of medical terms, and relations between terms. WIM agents include capabilities that can elaborate queries using terms from such a medical thesaurus. The

semantic relations contained in the thesaurus (synonyms, hyponyms and hypernyms, etc.) can be used by query elaboration capabilities to build new queries that are a generalization or a specialization of the original query.

- A collection of *source descriptions* is used to specify the search modes and filters allowed by each information source. In WIM information sources are accessed through agentified Web-based search engines, like Pubmed, which is used to access the Medline database, and Internet Grateful Med (IGM), which is used to access both Medline and HealthStar. A source description contains also a mapping between the bibliographical concepts used to generate queries at the conceptual level, and the concepts used by specific information sources. Source descriptions are used to customize queries for specific information sources.

- A collection of *categories* characterizing concepts from Evidence-based Medicine. This domain model characterizes a collection of *categories* in terms of keywords (terms) and filters that are useful to rank documents according to the EBM concepts. This knowledge is required by query elaboration capabilities such as the skill Query-expansion-with-category, to generate domain queries using the EBM category preferred by the user.

7.6.1 Evidence-Based Medicine

From the point of view of the Evidence-Based Medicine, it is very important to use the bibliographic references according to the quality of the evidence they rely on; hence, we have defined some categories expressing concepts about medical evidence quality in terms of a medical thesaurus and the defined ontology for bibliographic data, furthermore these concepts are weighted, allowing to weight the queries according to these "evidence quality" indicators.

Specifically, we have build 15 categories about EBM and some medical categories that are often required by medicine professionals using EBM. These categories belong to four different topics: *Evidence-Quality*, *Clinical Categories*, *Analysis* and *Evidence Integration*. See Figure 7.12 for a taxonomy of the medical categories defined in the EBM domain-model.

A category is defined as a structure of terms and filters associated to one topic, in which both filters and terms are weighted according to the strength of that association. Figure 7.7 shows the main concepts used to describe categories. For example *Guidelines* is a category that defines some filters to get only papers offering clinical advice based on good evidence quality. Table 7.6.1 shows the attribute-value definitions of the Guidelines category.

7.6.2 The MeSH thesaurus

A thesaurus is a book or a digital repository of synonyms, often including related and contrasting words and antonyms. There are general thesaurus concerning

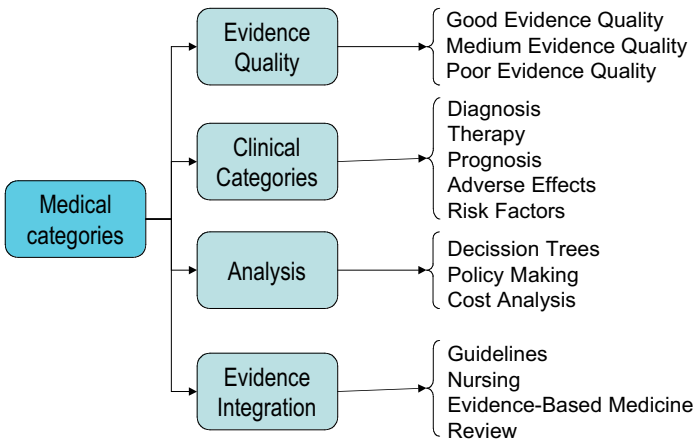


Figure 7.12: Categories on Evidence-based Medicine

Category	***Guidelines***	
Attribute	**Value**	**Weight**
Publication Type	Guidelines	1.0
Publication Type	Practice Guidelines	0.8
Publication Type	Guideline Adherence	0.6
Publication Type	Clinical Protocols	0.4

Table 7.1: Definition of the category *Guidelines*

MeSH Heading	Pneumonia
Tree Number	C08.381.677
Tree Number	C08.730.610
Scope Note	Inflammation of the lungs
Entry Term	Experimental Lung Inflammation
Entry Term	Lung Inflammation
Entry Term	Pneumonitis
Entry Term	Pulmonary Inflammation
Allowable Qualifiers	BL CF CI CL CN ...
Unique ID	D011014

Table 7.2: Definition of the term *Pneumonia* in Mesh

an entire language, but there are also specialized thesaurus containing concepts and vocabulary of a particular field, as of medicine or music.

The Medical Subject Headings (MeSH) is a controlled vocabulary produced by the National Library of Medicine and used for indexing, cataloging, and searching for biomedical and health-related information and documents. MeSH is the thesaurus used to index the bibliographic references stored in Medline, which is main database accessed by WIM information retrieval agents.

Table 7.6.2 shows an example of the information provided for the term *Pneumonia*, which is a MeSH heading. Each MeSH heading has a collection of synonyms specified as *Entry Terms* (e.g. Lung Inflammation, Pneumonitis and Pulmonary Inflammation), several *tree numbers* identifying the position of the term in the MeSH structure, and a collection of allowable modifiers (e.g. BL refers to *Blood*, used for the presence or analysis of substances in the blood; also for examination of, or changes in, the blood in disease states).

However, WIM capabilities can not use the MeSH information straightforwardly; instead, WIM capabilities are based on the ISA Ontology, which defines a term as having parent terms, children terms and other related terms specified as term-correlations. Therefore, we have implemented a mapping from MeSH headings to WIM terms so as to allow WIM capabilities to work with them. Entry terms become term-correlations with a correlation equal to one, which the maximum, because entry terms are synonyms of the main term (a MeSH heading). The *parents* (hyponyms) and the *children* (hypernyms) of a term are obtained from the MeSH structure using the tree number. For instance, *Pneumonia* has *Lung Diseases* as parent, and *Bronchopneumonia*, *Pleuropneumonia*, *Aspiration Pneumonia*, *Bacterial Pneumonia* and others as children. Although it is possible to download MeSH from the Web to use it locally, we have decided to access MeSH directly through the Web-based MeSH Browser, thus we have built a Wrapper to retrieve MeSH information from the Web and transform it into WIM terms. An example of a term is the following:

```
(define (term :id Pneumonia)
  (name "Pneumonia")
  (term-correlations (define (Term-Correlation)
                        (term Lung Inflammation)
                        (weight 1))
                     (define (Term-Correlation)
                        (term Pneumonitis)
                        (weight 1))
                     (define (Term-Correlation)
                        (term Lung Inflammation)
                        (weight 1))
                     (define (Term-Correlation)
                        (term Pulmonary Inflammation)
                        (weight 1))))
  (parent Lung Diseases)
  (children Bronchopneumonia
            Pleuropneumonia
            Aspiration
            Pneumonia
            Bacterial
            Pneumonia))
```

The idea of using term-correlations instead of synonyms is to allow more complex relations between words, like those provided by a Latent Semantic Analysis.

7.6.3 Medical sources

In order to facilitate the integration of multiple information sources, we have decoupled the knowledge on information sources from the query customization and retrieval capabilities. The idea of WIM is to provide a mediation service between the user putting a query, and multiple information sources. Such a mediation service must offer the user a single interface to many search engines, and aggregate information coming from heterogeneous sources. This type of mediation is often called Intelligent Information Integration (I3). As a result of reviewing many Information Integration systems (e.g. the Information Manifold [Kirk et al., 1995, Levy et al., 1996], TSIMMIS [Chawathe et al., 1994], Infomaster [Genesereth et al., 1997], Infosleuth [Nodine et al., 1999], and SIMS [Arens et al., 1993, Knoblock et al., 1994]), we concluded the importance of using a *lingua franca* to specify user level queries, and mapping schemas to transform concepts specified in the lingua franca into concepts used by specific information sources. In WIM, the queries expressed in the lingua franca are called *domain queries*, and the source-specific queries are called *source queries*.

On the one hand, we have elaborated a bibliographic-data ontology (Figure 7.13) encompassing the usual concepts found in the most popular bibliographical databases in medicine: Medline and Healthstar. Some of these concepts are the

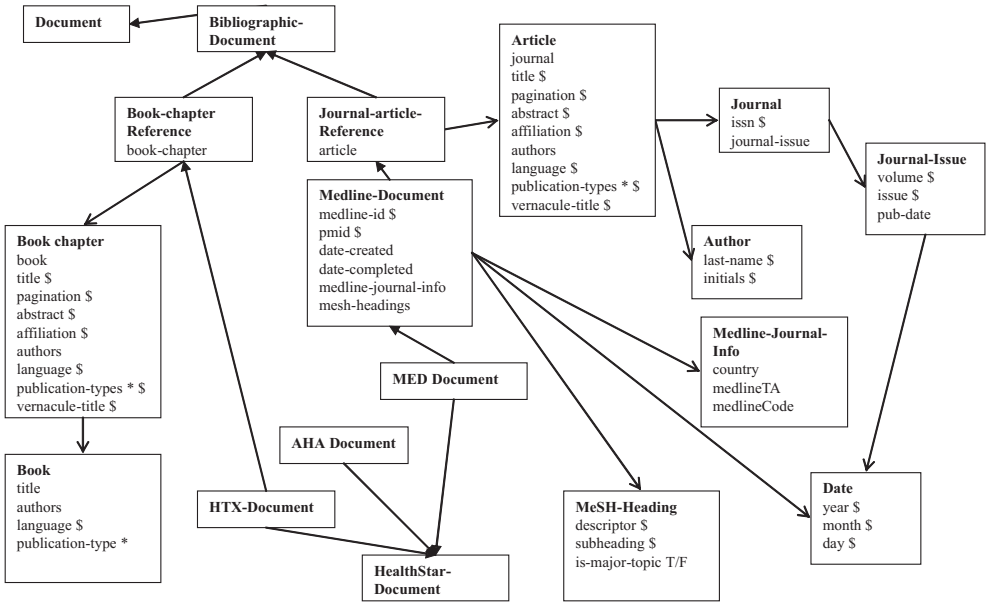


Figure 7.13: *Bibliographic-data* ontology

following:

- author name,
- publication type (e.g. journal, book chapter, proceedings, etc.),
- language,
- affiliation of the authors,
- begin year (lower-limit for publication date),
- end year (upper-limit for publication date),
- substance (drugs and pharmacologic terms),
- study groups (for statistically-based studies)
- age groups (for statistically-based studies)

On the other hand we have build an ontology to describe information sources . Recall that information sources in WIM are not databases as such, instead they are Web based search engines used to access the content within a database; for example, Pubmed is used to access Medline, and Internet Grateful Med is used to access Healthstar. There are two types of elements used to specify queries in Web-based search engines concerning bibliographical databases (Figure 7.8): search modes and search filters. Search attributes refer to different search modes

allowed by a search engine (e.g. searching a keyword only at the title, the abstract or both); moreover, we distinguish between the basic or default search mode (*basic attribute*), and other search modes supported by a search engine (*search-attributes*). Filters are used to constrain the search, using bibliographic concepts from the subsect of bibliographic concepts selected.

In WIM search modes are not specified by the user, they are incorporated into a query by the query customization capabilities in order to assess the results (§7.2) of a query. Therefore, only the filters require a mapping schema definition to translate the filters specified by the user (domain attributes) to equivalent filters expressed in a source-specific vocabulary (source attributes). An example of a source description is included in §7.7.2.

7.7 Exemplification of the WIM library

We have already introduced the domain knowledge used in the WIM application: the MeSH thesaurus, the EBM categories and the information sources encompassing bibliographical references about medicine. In this section we will show by means of an example how the domain knowledge is used to generate queries from a user consultation, and how the results are aggregated and ranked with respect to the relevance and utility of the user consultation. Section 7.7 shows how WIM configures a team for a specific problem, from the problem specification stage to the solution of the problem by a team of agents.

7.7.1 Query Elaboration using EBM

Suppose the user wants to find information about *guidelines* on the use of *levofloxacin* in the treatment of *pneumonia*. Therefore, the selected keywords should be the name of the substance (levofloxacin), the disease (pneumonia), and the type of information searched (guidelines). In addition, the user decides to limit the search to articles published after 1980, thereby he specifies a filter for the attribute *Begin Year* with a value of 1980. The consultation is equivalent to the the following query

```
query.terms: levofloxacin, pneumonia, guidelines
query.filters: (Begin Year, 1980)
```

The agent applying the Query-expansion-with-categories skill takes the knowledge on the Guidelines category (see Table 7.6.1) from the EBM knowledge base. That agent find that the Guidelines category is related (with a specific correlation strength) to the following terms: "practice-guidelines" (with strength of 0.8), "guideline-adherence" (with strength 0.6), and "clinical-protocols" (with strength 0.4).

The system uses term correlations to generate new *query models* where the original keywords are replaced by the correlated terms. The output of the capability is a collection of query models with weights corresponding to the correlation strength of the term being used. Only one term is replaced by a correlated

term at each output query. In the example, the generated query models have a domain query whose terms are the same terms specified as keywords in the user's query except for "guidelines", that is replaced by one of their correlated terms —practice-guidelines, guideline-adherence and clinical-protocols.

```
Query model 1 Query.terms:
levofloxacin pneumonia guidelines Query.filters: (Begin Year,
1980) Weight: 1

Query model 2 Query.terms: levofloxacin pneumonia
practice-guidelines Query.filters: (Begin Year, 1980) Weight: 0.8

Query model 3 Query.terms: levofloxacin pneumonia
guideline-adherence Query.filters: (Begin Year, 1980) Weight: 0.6

Query model 4 Query.terms: levofloxacin pneumonia
clinical-protocols Query.filters: (Begin Year, 1980) Weight: 0.4
```

7.7.2 Basic Query Customization

Once the Elaborate-query task is finished, and before the start of the Customize-query task, some information sources are selected from a set of allowable sources. For simplicity, let's assume the two sources have been selected: *Pubmed* and *HealthStar*. Then, the capability selected to solve the Customize-query has to adapt the domain queries obtained by the Query-expansion-with-categories capability to those two information sources.

The Query-customization capability expands a query expressed in a source independent way (a domain query) into a collection of queries in terms of a particular information source (source queries), using the search modes and filters allowed by each source. The properties of this knowledgeare characterized by a domain-model called Source-Descriptions. The source-description for the Pubmed information source (a Web-based retrieval engine to the Medline database) is included below as an example.

```
(define (source :id Pubmed)
 (name "PubMed")
 (weight 1)
 (search-attributes
                      (define (Attribute-Weighting)
                        (attribute "MAJR")
                        (weight 1))
                      (define (Attribute-Weighting)
                        (attribute "MH:NOEXP")
                        (weight 0.8))
                      (define (Attribute-Weighting)
                        (attribute "MH")
```

```
                        (weight 0.6))
                      (define (Attribute-Weighting)
                        (attribute "TI")
                        (weight 0.5))
                      (define (Attribute-Weighting)
                        (attribute "TW")
                        (weight 0.4)))
  (basic-attribute (define (Attribute-Weighting)
                        (attribute "ALL")
                        (weight 0.2)))
  (filter-attributes
                      (define (Attribute-Translation)
                        (domain-attribute "Author Name")
                        (source-attribute "AU"))
                      (define (Attribute-Translation)
                        (domain-attribute "Publication Type")
                        (source-attribute "PT"))
                      (define (Attribute-Translation)
                        (domain-attribute "Language")
                        (source-attribute "LA"))
                      (define (Attribute-Translation)
                        (domain-attribute "Affiliation")
                        (source-attribute "AF"))
                      (define (Attribute-Translation)
                        (domain-attribute "Journal")
                        (source-attribute "JO"))
                      (define (Attribute-Translation)
                        (domain-attribute "Begin Year")
                        (source-attribute "MINDATE"))
                      (define (Attribute-Translation)
                        (domain-attribute "End Year")
                        (source-attribute "MAXDATE"))))
```

The Query-customization capability (a skill) performs two kind of transformations for each pair consisting of a query and a source:

- *from domain filters to source filters*: the query filters, written in terms of the domain terminology (bibliographic data) are rewritten in terms of the source terminology, according to the attribute-translations in the source domain-model; and
- *from keywords to search-attributes*: each search term is transformed into a filter where the value is the term and the attribute is one of the search-attributes allowed by the source. These filters are called *t-filters* in the ISA-Ontology, though they are objects of sort Filter.

A new query model is created using the *basic-attribute* to build the *t-filters*. The *filters* are translated using the translation rules defined in the Source De-

scriptions. For example, for the Query Model 1 and the source Pubmed, the following transformation is applied.

```
Query Model 1 before the customization:
    Query.terms: levofloxacin pneumonia guidelines
    Query.filters: (Begin Year, 1980)
    Weight: 1

Query Model 1 after the customization:
    Query.t-filters: (ALL, levofloxacin), (ALL, pneumonia),
                     (ALL, clinical-protocols)
    Query.filters: (MINDATE, 1980)
    Source: PubMed-Medline weight: 0.2
```

Note that all the *t-filters* use the basic attribute (called "ALL") and the attribute "Begin Year" is replaced by MINDATE.

A new query model is created for each *term* and each *search-attribute* in the source domain-model. The following query models are obtained when using the PubMed *search-attributes* (TW, TI, MH, MH:NOEXP and MJR) in place of the basic-attribute (ALL), for the first term.

```
Query model 1.2.a
    query.t-filters: (TW, levofloxacin), (ALL, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 0.4

Query model 1.3.a
    query.t-filters: (TI, levofloxacin), (ALL, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 0.5

Query model 1.4.a
    query.t-filters: (MH, levofloxacin), (ALL, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 0.6

Query model 1.5.a
    query.t-filters: (MHNOEXP, levofloxacin), (ALL, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 0.8
```

```
Query model 1.6.a
    query.t-filters: (MAJR, levofloxacin), (ALL, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 1
```

In the example above each *query model* is identified by two numbers and a letter separated by dots: QM x.y.z identifies the y^th *query-model* (containing a source query) obtained for the x^{th} *query-model* (containing a domain query) and source z (a for Pubmed and b for IGM-HealthStar). The weights of the resulting *query-models* are calculated multiplying the weight of the input *query-model* by the weight of the *search-attribute* used, or multiplying by the weight of the *basic-attribute* if no *search-attribute* is used (e.g. QM 1.1.a).

Modifying the *search-attribute* for the second or third *term* and keeping the other terms associated to the *basic-attribute* results in 10 new query models, 5 per term (QM 1.7a – 1.11a and 1.12a – 1.16a).

```
Query model 1.7.a
    query.t-filters: (ALL, levofloxacin), (TW, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 0.4
    ...

Query model 1.11.a
    query.t-filters: (ALL, levofloxacin), (MAJR, pneumonia),
                     (ALL, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 1
    ...

Query model 1.16.a
    query.t-filters: (ALL, levofloxacin), (ALL, pneumonia),
                     (MAJR, guidelines)
    query.filters: (MINDATE, 1980)
    weight: 1
```

At the end, each domain-query has originated 16 source-queries when customized for PubMed; therefore, since the user consultation has produced 4 *domain-queries* (QM1a, QM4), a total of 4*16 = 84 source-queries are obtained for PubMed. The same *domain-queries* are customized for HealthStar, which has only *search-attribute* different from the *basic-attribute*, therefore only 4 source queries are generated for each domain-query, summing 16 queries in total.

Finally, the 84 Pubmed queries and the 16 Healthstar queries are sent to the wrappers to each information source, which perform the final transformation of the queries to fit the specific interface of each information source.

Let us briefly explain the operation of a wrapper, for instance the Pubmed-wrapper. The wrapper concatenates at least three strings: the server URL, a path to the source location in the server and the query itself, according to the source query format. For example, for the Pubmed-wrapper and the Query Model 1.6.a:

```
Server URL: "http://www.ncbi.nlm.nih.gov/"
Path: "entrez/query.fcgi?"
query.t-filters: (MAJR, levofloxacin), (ALL, pneumonia),
                                       (ALL, guidelines)
query.filters: (MINDATE, 1980)
```

The resulting URL is

```
entrez/query.fcgi?CMD=search\& DB=PubMed\&term=
levofloxacin[MAJR]+AND+pneumonia[ALL]+AND+
guidelines[ALL]+AND+1980[MINDATE]
```

The wrapper also deals with the problem of the format in which results are retrieved, the number of items to retrieve and other technical issues.

7.7.3 Aggregation

When a weight is assigned to a query after applying some transformation, this is expressing the relative importance or representativity of that query with respect to the original one. The meaning of a weight assigned to a query is logically inherited by the documents or items retrieved for that query, thus we can say that the weights associated to the items retrieved represent the membership of those elements to the topic requested by the user. Aggregation operators for numeric values can be used here. The most widely used operators are the *arithmetic mean* and the *weighted mean*, but there is a family of aggregation operators, including fuzzy measures; WIM implements several operators reviewed in [Torra, 1996].

Let's see an example. First, suppose that Pubmed has been queried using four queries with the following weights: 1, 0.8, 0.6, and 0.4. Suppose that one of the retrieved items appears in queries 1 and 2, but not in 3 and 4. As PubMed does not rank documents, WIM assigns by default a maximum score of one to both occurrences of the same item. When doing the aggregation, absence of items are also taken into account; each absence is represented like a "presence" with a value equals to 0, as showed below.

Query	w_i	$\bar{w}_i$	$score_i$
Q_1	1,0	0,36	1
Q_2	0,8	0,29	1
Q_3	0,6	0,21	0
Q_4	0,4	0,14	0

Before any aggregation, the weights of the queries have to normalize, because aggregation operators assume that $w_i \in [0,1]$ and $\sum_{\forall i} w_i = 1$. We denote this

normalized weight as $\bar{w}$. We can apply for instance a weighted-mean operator, defined as $WM(Q_1, \ldots, Q_n) = \sum_{i=1\ldots n} \bar{w}_i \cdot score_i$, and the aggregate value is 0,65.

After finishing the aggregation the user gets the results of his consultation; some examples of the *scored-items* obtained are shown below:

```
Scored item 1
    identifier: PMID10743984
    title: Economic assessment of the community-acquired pneumonia
    intervention trial employing levofloxacin.
    score: 0.187

Scored item 2
    identifier: PMID10683053
    title: A controlled trial of a critical pathway for treatment
    of community-acquired pneumonia
    score: 0.172

Scored item 1
    identifier: PMID11557471
    title: Pharmacodynamics of fluoroquinolones against
    Streptococcus pneumoniae in patients with community-acquired
    respiratory tract infections
    score: 0.062
```

7.8 Experimental results

We have developed two applications where the query weighting approach has been used: MELISA and WIM.

MELISA [Abasolo and Gómez, 2000] is a single agent to look for medical literature, which is based on the use of ontologies to separate the domain knowledge from source descriptions. The notions of query weighting and exploiting the filtering capabilities of existing search-engines are already used here, but only one search-engine is included. Although the aggregation procedures were not systematized yet, the system demonstrated that the query weighting and aggregating framework provides an accurate and flexible approach to rank documents. MELISA has proven the flexibility of this framework to apply utility-criteria to rank documents. In particular, a collection of knowledge categories has been used to rank medical references according to the quality of the evidence they are based on, which is called Evidence Based Medicine [Feinstein and Horwitz, 1997]. The query weighting process is applied twice, one at the domain level and the other at the source level. Query weighting at the domain level is carried over using knowledge categories; each category is a medical topic described as a collection of weighted elements. The elements used to describe medical categories are medical terms, used as keywords, and other search attributes used to look for

bibliographic references, like publication type, related terms, and so on. Query expansion at the source level is achieved by using the search modes described in §7.7.2 for the Pubmed source.

We have compared the results of MELISA (using PubMed as the search engine) against those results obtained with the PubMed for some user queries. An expert has proposed us 5 cases to look for medical references based on his everyday work. Every case has been translated into a query suitable for both PubMed and MELISA. The results have been scored by two different evaluators, instructing them to score the references according to their degree of relevance for the corresponding query and the quality of the evidence. The first 40 references retrieved for each query have been evaluated by the experts. An ordinal scale with three levels has been used to score references: 2 for references that satisfy the user's need, 1 if the reference is simply related, 0 if it is not relevant, and "?" if the expert has no enough information (the expert has only the title and abstract of the documents) about a bibliographic reference to asses its relevance.

Table 7.3 sums up the results of the assessment procedure for the relevance. As you can see in the table, the results obtained by MELISA have been evaluated as being more relevant than those obtained by querying Pubmed directly. Specifically, there are more bibliographic references considered relevant for the user's need, and fewer references ranked irrelevant or without enough information to be evaluated.

Relevance	PubMed	MELISA
2	38%	48%
1	16%	17%
0	13%	9%
?	34%	27%

Table 7.3: Assessment of relevance

The utility of the proposed framework to develop and use new ranking criteria has also been tested. Some of the implemented medical categories are specifically designed to represent notions about evidence quality in medical references. Using these categories to enrich queries during the query expansion has demonstrated that the resulting references are well ranked according to the quality of the evidence. The testers have been instructed to classify references into three levels: good, medium and poor evidence. The symbol "?" is used to note references without enough information to be evaluated according to the quality of the evidence.

Table 7.4 sums up the results of the assessment procedure concerning the quality of the evidence. As you can see in the table, results are not so easy to interpret as for the relevance. Notice that there are a greater percentage of items assessed as being based on a good evidence quality (20% to 7%), though surprisingly there are also more items that have been assessed as backed with a poor evidence quality (9% to 2%). Another datum to be remarked is the very high percentage of items with not enough information. The point is that we

are working with the bibliographic references only, and not with the documents (mainly articles) being referenced, thus usually there is no enough information to assess the quality of the evidence a referenced document is based on.

Evidence	PubMed	MELISA
Good	7%	20%
Medium	2%	3%
Poor	2%	9%
?	89%	69%

Table 7.4: Assessment of evidence quality

To sum up, MELISA has proven the utility of the information search approach adopted in WIM to rank documents according to utility criteria like EBM quality: in our experiments, MELISA has received a stronger approval assessment from expert user than PubMed.

However, we expect WIM to obtain similar or even better results than MELISA, since WIM extends the core functionality implemented in MELISA. The reason is that MELISA implements a fixed combination of query elaboration, customization and aggregation methods, whilst WIM allows for a multiplicity of tasks and methods that can be combined in different ways through the Knowledge Configuration process. Moreover, WIM is configured on-demand according to problem requirements, thus the resulting agent team is tailored to the problem at hand to better fit the needs and preferences of the user. The present evaluation, assesses just the core functionality of the new information search techniques implemented in WIM's library.

7.9 Example of the Cooperative Problem Solving process in WIM

This section illustrates the use of the ORCAS framework in the WIM application to carry out an information search task by customizing a team of problem solving agents on-demand, according to the ORCAS model of the Cooperative Problem Solving process. We show a complete example on how WIM operates, following the different stages of the CPS process as implemented in the ORCAS institution, which encompasses the following processes: Registering of capabilities, Brokering (Knowledge Configuration), Team Formation and Teamwork. In addition, the Problem Specification process is included to show the way a problem is specified by a user or an external agent.

The WIM example is illustrated with some snapshots from the Agent World, a 3D World environment that represents the agent communication and agent flow between scenes in the e-institution (external view), together with information on agents state (internal view). The Agent World external view shows an e-Institution as a set of rooms representing the scenes (big rectangles), and corridors representing transitions between scenes (little rectangles connecting one

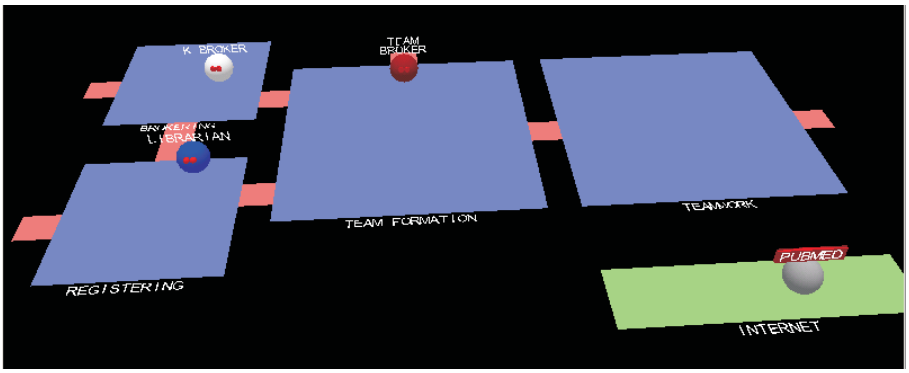


Figure 7.14: Overview of the ORCAS e-Institution

scene to another). Agents are represented as eyed-balls staying at a room or moving between rooms through transitions, and the messages they send are represented as little balls colored according to the type of message (e.g. request, inform, etc.).

Image 7.14 shows a zoomed out view of the ORCAS e-Institution in the Agent World tool. One can see the main scenes of the ORCAS e-Institution and the allowed transitions between scenes, together with some institutional agents waiting for other agents. At the Registering scene there is a Librarian agent waiting for new Problem Solving Agents (PSAs) willing to join the institution; at the Brokering scene there is a Knowledge-Broker waiting for a Personal Assistant (PA) having a problem to be solved; and at the Team Formation scene there is a Team-Broker waiting for both a PA willing to form a team of agents, and a set of PSAs providing their capabilities.

7.9.1 Registering capabilities

In order to become available to other agents, PSAs must register their capabilities to a Librarian agent, according to the specification of the Registering scene. Capabilities are registered using the ORCAS Knowledge Modelling Ontology and Feature Terms as the Object Language.

Figure 7.15 shows a screenshot of the Registering scene. After registering their capabilities to the Librarian, the PSAs move to the Team Formation scene to wait for team-role proposals to joining a team. At the moment captured, the PSA Marteen has just finished the Registering scene and is moving to the Team Formation scene.

The Registering scene is instantiated once for each new PSA entering the institution, but the Agent World monitoring tool shows only one scene that is a synthesis of all the instances of the scene occurring simultaneously; therefore, several PSAs can be shown in the Registering scene at a time.

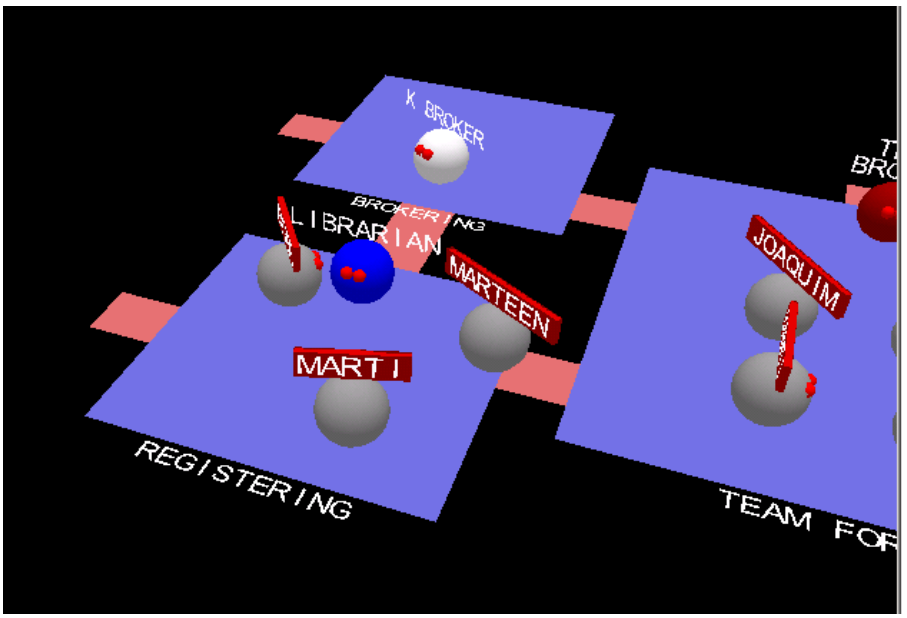


Figure 7.15: Screenshot of the Registering scene

The Librarian keeps an updated register of the capabilities registered by the PSAs that have joined the system. Figure 7.16 shows a screenshot of the Librarian internal state displaying the record of capabilities registered in the institution by WIM PSAs.

7.9.2 Problem specification

The WIM application can process requests coming from different sources: from a windows-like client, from Web pages, and from external agents using either the NOOS Agent Communication Language (ACL) or the FIPA-ACL (§7.3 describes the overall architecture). We will show the way a problem is specified by the end user and the way it is encoded in agent terms.

A problem is specified by input data and problem requirements, as described in §4.4.2. The input data for an information search task in WIM is a consultation, composed of a set of keywords and filters plus a set of information sources and optionally a knowledge category. We are going to show first how a problem is specified using the WIM panel, and the way a problem is specified using the Agent Communication Language afterwards.

Figure 7.17 shows the consultation screen where the user specifies a consultation, including just a portion of all the filters allowed by the WIM application (see §F.3). In the example, the user has introduced three keywords (ofloxacin, pneumonia and guidelines) and a filter (from year 1980). The user has a limited control over the type of task to be solved, either the task Information Search

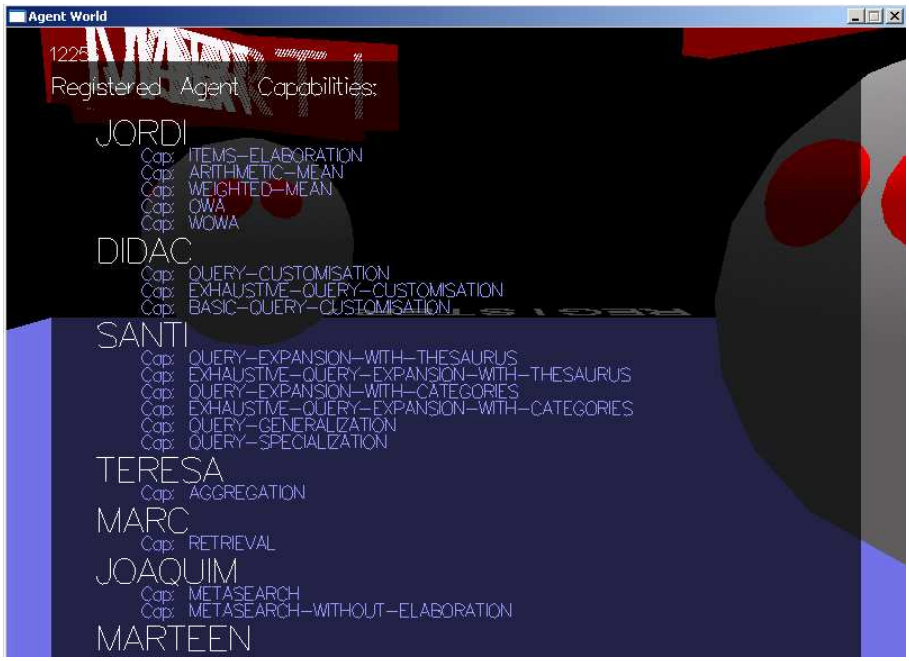


Figure 7.16: Example of the Librarian internal state

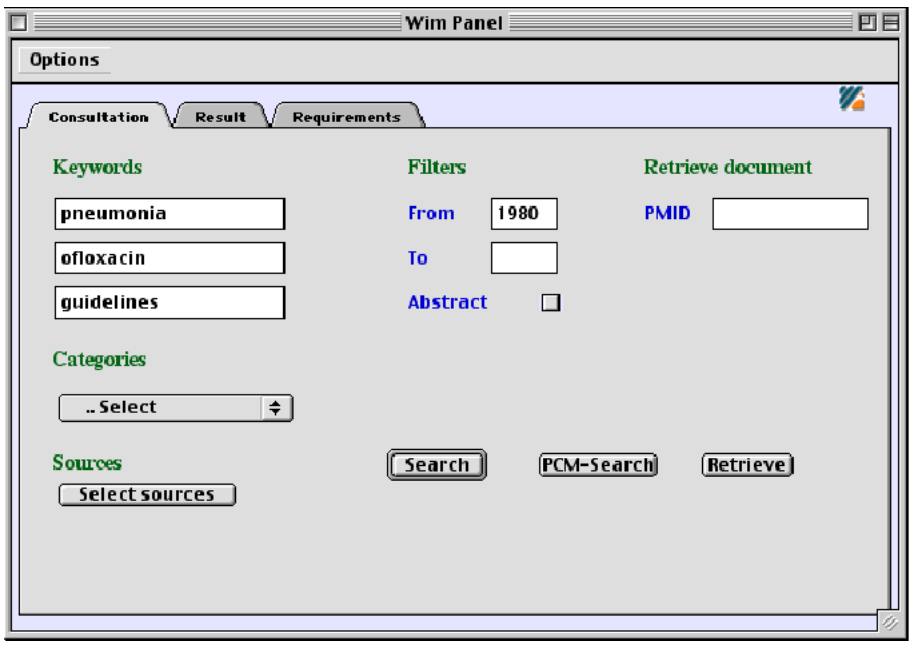


Figure 7.17: Consultation example

(buttons labelled `Search` and `PCM-Search`) and the task Retrieve-full-reference (button labelled `Retrieve` in the figure). Furthermore, the user can decide whether to apply the task-decomposer Metasearch or the task-decomposer PCM-Search to solve the Information Search task; which is discriminated by pushing either the button labelled `Search` or the one labelled `PCM-Search`.

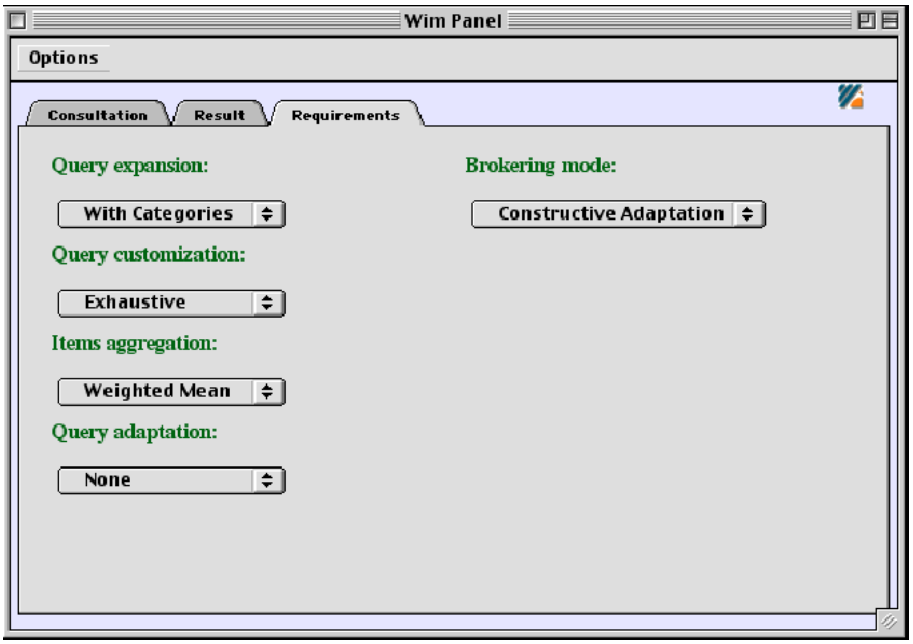


Figure 7.18: Consultation example

Figure 7.18 shows the screen where the user specifies the problem requirements. That screen is an interface where the user chooses some among a limited number of requirements organized according to the tasks they are related with (as a curiosity, compare it with the interface showed in 4.23, which is addressed to the knowledge/software engineer and offers all the requirements available in the library following a plain, quite raw style).

In our example, the user has selected some options corresponding to the following postconditions:

- Elaborate-query-with-thesaurus
- Non-exhaustive-customization
- Aggregate-with-arithmetic-mean

Each option corresponds to a postcondition added to the problem requirements to discriminate among alternative capabilities for the same task. Other requirements are added automatically by the GUI depending on the button pushed

by the user. For instance, if the user pushes the button labelled `PCM-Search`, two extra postconditions will be added to the requirements to ensure that the capability PCM-Search will be selected during the Knowledge Configuration process instead of Metasearch:

- Satisfy-consult
- Asses-search-results

Moreover, the options screen allows the user to select either the *Search and Subsume* or the *Constructive Adaptation* strategy (see §4.4.4) for the Knowledge Configuration process.

Once a problem has been specified using whatever means, it should be encoded in agent understandable terms, using either the NOOS ACL or the FIPA-ACL. We show below a FIPA-ACL specification of a message requesting the WIM PA to perform a Cooperative Problem Solving process according to the requirements stated above, assuming the user has pushed the button labelled `PCM-Search`.

The request message includes both the problem requirements and the problem data corresponding to the former example, encoded in XML within the content of the message. Notice that the XML encoded data is embedded within a FIPA-SL0 envelop defining an "action". See Appendix F for a more detailed account of the ORCAS services.

```
(request
    :sender     (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
    :receiver   (set (agent-identifier :name \wim-PA@wim.iiia.csic.es:7778/NOOS))
    :reply-with     cps-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    cps18236
    :content
       (action
        (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
        (Cooperative-Problem-Solving
           :problem-requirements
              (Problem-Requirements
                 :encoding  <xml? version="1.0" encoding ="ISO-8859-1"?>
            :value
               <problem-requirements>
               <task-name>PCM-Search</task-name>
               <postconditions>
                  <formula>Elaborate-With-Thesaurus</formula>
                  <formula>Non-Exhaustive-Customization</formula>
                  <formula>Aggregate-With-Arithmetic-Mean</formula>
                  <formula>Satisfy-Consult</formula>
                  <formula>Assess-Search-Results</formula>
               </postconditions>
               <input-roles>
                  <signature-element>Query-Model</signature-element>
               </input-roles>
               <domain-models>
                  <domain-model>Medical-Sources</domain-model>
                  <domain-model>MeSH</domain-model>
                  <domain-model>EBM</domain-model>
               </domain-models>
```

```
    </problem-requirements>)
:problem-data
   (User-Consult
      :encoding  <xml? version="1.0" encoding ="ISO-8859-1"?>
  :value
     <user-consult>
     <query>
     <keywords>
        <keyword>Ofloxacin</keyword>
        <keyword>Pneumonia</keyword>
        <keyword>Guidelines</keyword>
     </keywords>
     <filters>
         <filter>
             <attribute>Begin-year</attribute>
             <value>1980</value>
         </filter>
     </filters>
     </query>
     <sources>
        <source>Pubmed</source>
     </sources>
     </user-consult>))))
```

The application task is Information-Search, and there are five postconditions in total, the first three ones specified by the user, and the other two added by the GUI, as explained above. Notice there are three domain-models added to the problem specification: Medical-sources, that provides a knowledge-role of sort Information-Sources, MeSH, that provides a knowledge-role of sort Thesaurus, and EBM, that provides a knowledge-role of sort Categories. Recall that the knowledge to be used by the selected capabilities must conform to the properties of the domain-models specified within the problem requirements (see §4.4.2 or §4.4.5). Therefore, all the domain-models available in WIM are included in the request; only the capabilities which knowledge-requirements are satisfied by one of these domain-models are potential candidates to be selected during the Knowledge Configuration process.

The Personal Assistant is able to parse messages specified in FIPA-ACL, extract the embedded data and translate them into the NOOS ACL before communicating with other agents to carry out the requested action, performing a Cooperative Problem Solving process throughout. To do that, the PA must participate in three different scenes within the ORCAS e-Institution: first, the PA goes to the Brokering scene to request the Knowledge-Broker for a task-configuration satisfying the problem requirements; second, the PA moves to the Team Formation scene and request the Team-Broker to form a team according to the just obtained task-configuration; and third, the PA moves to the Teamwork scene and solicits the team-leader to solve the problem using the specified problem data.

7.9.3 Knowledge Configuration

The Knowledge Configuration process has the goal of finding a configuration of agent capabilities satisfying the requirements of a problem to be solved. These requirements are specified as signatures (inputs, outputs and domain-models)

and formulae (preconditions and postconditions), represented with the same language that the agent capabilities, thus allowing to match the problem specification against the registered agent capabilities.

Recall that the Knowledge Configuration process is performed by the Knowledge-Broker (K-Broker) at the Brokering scene. The Brokering scene (§6.5.2) starts with the PA sending a problem specification to the K-Broker to request for a task-configuration satisfying that specification. Figure 7.19 shows the Brokering scene when the PA has just send the *request* message (represented by a small ball in the figure) to the K-Broker.

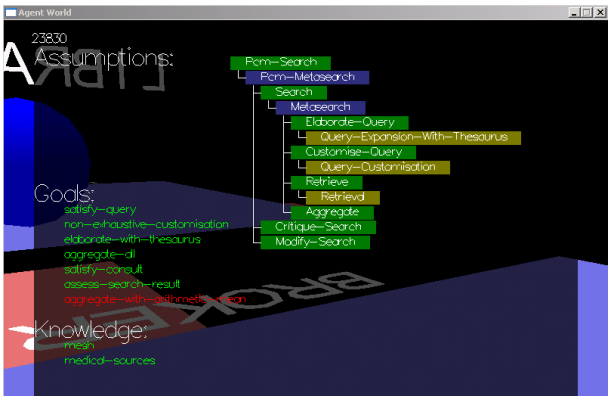


Figure 7.19: Screenshot of a Brokering scene

Following, the K-Broker asks the Librarian to get an up-to-date version of the tasks and capabilities available in the system, and after that, the K-Broker starts a Knowledge Configuration process over those specifications.

Figure 7.20 shows a screenshot of the K-Broker internal state some moment before finding a task-configuration. The left side of the screen shows the assumptions and postconditions (called goals), both the already satisfied ones (green/light gray) and the ones to be yet satisfied (red/dark gray), plus the selected domain models (knowledge). The right side of the screen shows the ongoing task-configuration.

Notice that there are more postconditions than those specified by the problem requirements, like SATISFY-QUERY and AGGREGATE-ALL. These postconditions have been introduced to the search state as a result of the tasks having been added during the Knowledge Configuration process by a task-decomposer. For instance, the postcondition SATISFY-QUERY has been added by the task Retrieve.

The task Information-Search has been bound to the task-decomposer PCM-Metasearch, which introduces three new tasks: P-Search, C-Search and M-Search. The task P-Search is bound to the capability Metasearch, which introduces four subtasks: Elaborate-query, Customize-query, Retrieve and Aggregate.

The task Elaborate-query is bound to the skill Query-expansion-with-thesaurus, because this is the only capability satisfying the user specified postcondition

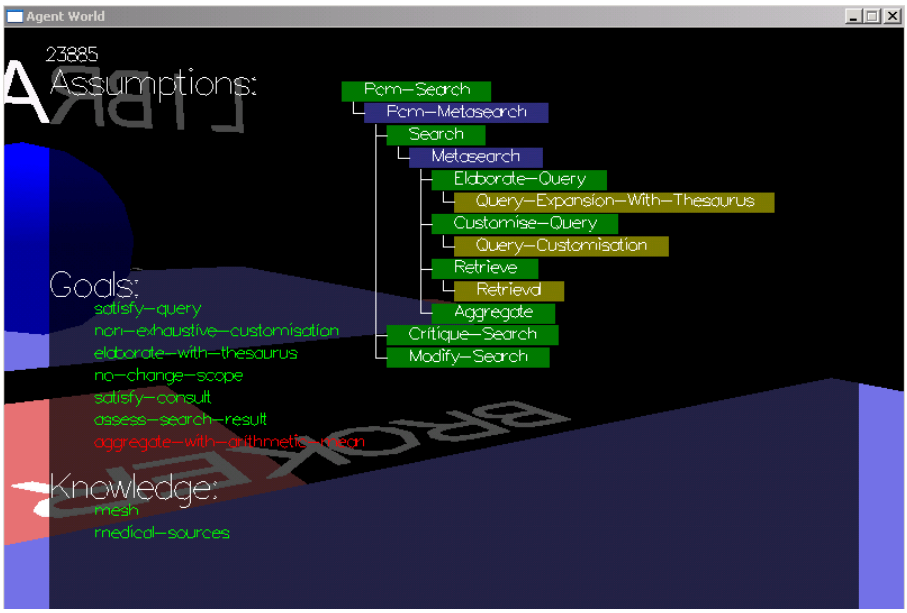


Figure 7.20: Example of the K-Broker internal state

ELABORATE-QUERY-WITH-THESAURUS. Customize-query is bound to Query-customization, due to the postcondition NON-EXHAUSTIVE-CUSTOMIZATION. Retrieve is bound to Retrieval, which is the only capability matching that task. However, Aggregate is not bound to a capability yet, and in accordance with it, the postcondition AGGREGATE-WITH-ARITHMETIC-MEAN has yet to be satisfied.

Figure 7.21 shows the final task-configuration for the current example. Notice that the task M-Search is not being configured because it depends on the result of the task P-Search, or more specifically, it depends on the result of the C-Search task, which has the purpose of assessing the result of P-Search. This example corresponds to the *Delayed Configuration* mode described in §5.7.1.

Finally, the K-Broker will send the PA an "inform" message containing the task-configuration in Figure 7.21, and after that the PA will move to the Team Formation scene where it will request the T-Broker to form a new team according to that task-configuration.

7.9.4 Team-formation

Team Formation is decomposed into three different activities: task allocation, team selection, and team instruction (§6.5.3). All these activities are coordinated by an agent playing the Team-Broker (T-Broker) role, following an auction-like interaction protocol.

Figure 7.22 captures the moment when the Team-Broker is sending team-role proposals to the available PSAs (e.g. Santi, Didac, Jordi), which will answer

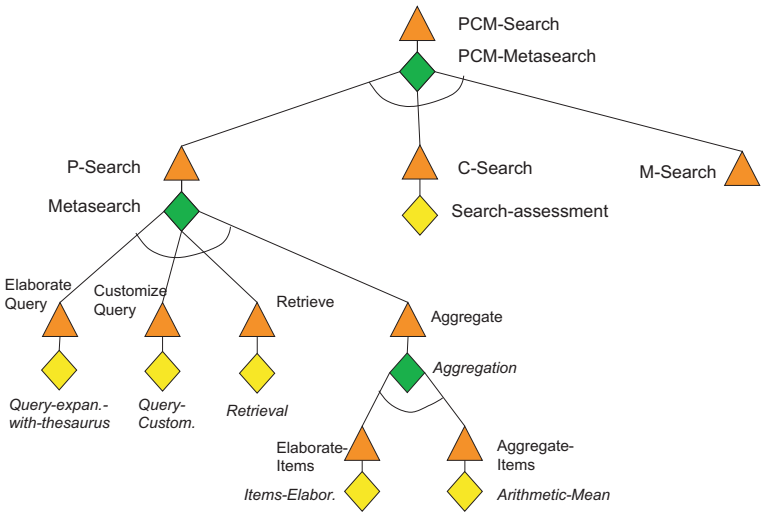


Figure 7.21: Task-configuration with a task in delayed configuration mode

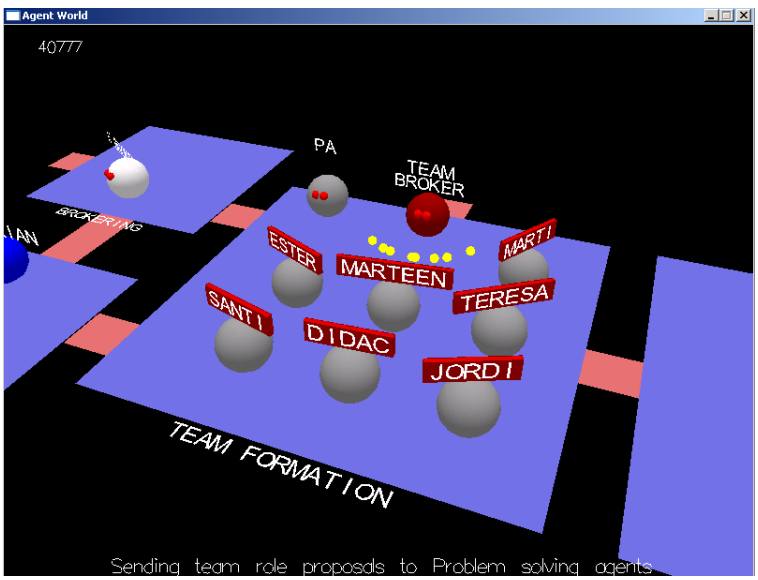


Figure 7.22: Screenshot of a Team Formation scene

whether they accept or refuse such a proposal. PSAs accepting a team-role proposal are considered candidate agents for the corresponding task (recall there is one team-role per task) by the T-Broker.

The T-Broker sends proposals until all the tasks are allocated to some agent or some failure criterion is reached that leads the T-Broker to abort the process without having succeeded. After that, if the task-allocation succeeded, the Team-broker chooses among alternative agents for the same task, and inform selected agents of the result.

Figure 7.23 shows a screenshot of the T-Broker internal state after having obtained candidates for all the tasks. The tasks to be allocated are on the left, while candidate agents are represented on the right. Agents already selected for a task are recognized by a square surrounding them. At the moment shown, the T-Broker has already finished the task-allocation process and is performing the team selection process (selecting agents for each task); for instance, Ester has been just selected for the task Retrieve, to the detriment of Marc, who is also a candidate agent for that task. Notice that the team selection activity proceeds following a bottom-up direction, from the tasks in the leaves of the task-configuration tree upwards, until reaching the top level task. Tasks with delayed configuration, such as M-Search, are not being allocated, since the capabilities to be applied for those tasks are not known yet.

Figure 7.23: Example of the T-Broker internal state

After finishing the team selection process, the T-Broker informs the participating agents on the result of the team selection process, and instructs selected agents about the team-roles they will play. A team-role (§5.3) includes the following information: a task to be carried out, the capability required to solve that task, and all the information required to cooperate with other team mates, which is encompassed by a collection of team-components. The agents obtained as candidates for some task but not selected in the end, are kept in reserve to replace a team-member if it fails during the Teamwork process, just in case.

At the end of the Team Formation scene, the T-Broker sends the resulting *team-configuration* to the PA, which is then ready to initiate the Teamwork scene. Meanwhile, the PSAs selected to participate in the team move to the Teamwork scene (in Figure 7.25 one can see the agent Marc waiting in the Team Formation scene, while all the selected agents have already moved to the Teamwork scene).

7.9.5 Teamwork

The Teamwork scene starts when the PA sends the data of the problem to be solved to the agent assigned the top-level task in the task-configuration, the so called team-leader. In the example, the team-leader is Marteen, who is assigned the task Information-Search, and has to apply the task-decomposer PCM-Metasearch.

The team-leader starts the problem solving activity using the information provided by the team-role for the top-level task, Information Search in our example. The information about which specific capability the team-leader has to apply for solving that task, and the name of the agents to whom delegate a task are described by a *team-role* provided to the team-leader by the T-Broker at the Team Formation scene. Besides this, any agent does the same when requested to play a team-role: consulting the information provided by the R-Broker and applying the specified capability.

Figure 7.24 shows a screenshot of a PSA internal state, indicating the capabilities it is equipped with and the team-roles it has to play, represented by pairs composed of a task and a capability bound to it. If a PSA has to apply a task-decomposer, the information contained in the team-role includes also the subtasks introduced by the task-decomposer and the agents to whom delegate each subtask. Figure 7.24 shows Marti's internal state, which is equipped with three capabilities: Metasearch, Metasearch-without-elaboration, and Modify-Metasearch. Marti has been assigned the team-role associated to the task P-Search, which is bound to Metasearch, a task-decomposer. The figure shows also the names of the agents assigned to each of the subtasks introduced by the task-decomposer Metasearch: Teresa is assigned to Aggregate, Ester to Retrieve, Didac to Customize-query, and Santi to Elaborate-query.

A Problem-Solving Agent (PSA) can form part of different teams at the same time, thus when a PSA receives a request to carry out the task associated to a specific team-role, it has to check whether it is assigned to that team-role and which capability to apply. Next, the PSA checks whether that capability is a skill or task-decomposer: if it is a skill the PSA applies that skill to solve the task and sends the result to the requester; otherwise the capability is a task-decomposer and the PSA will follow the task-decomposer's operational description.

During the Teamwork scene, the control flow follows the performative structure specified by the operational description of the task-decomposers in the task-configuration, as described in §5.6.

Figure 7.25 shows a picture of the teamwork scene, just when the PSA assigned to the retrieve task is sending queries to the Pubmed information source

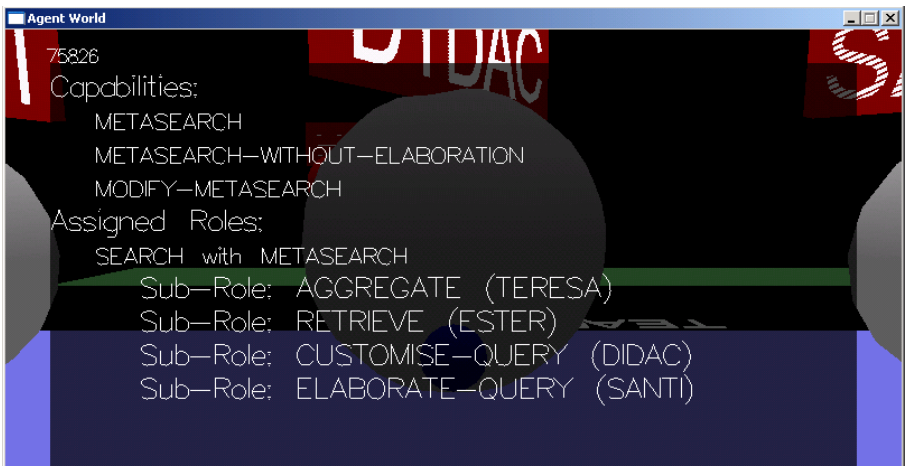


Figure 7.24: Example of a PSA internal state

through the Internet, using the Pubmed wrapper.

If all tasks are solved successfully, the final result is hold by the team-leader, who sends it to the PA, and the Cooperative Problem Solving finishes.

In our example, there is a task that is not bound to any capability because this decision depends on the result of another task, Critique-Search. Consequently, the agent detecting that condition (the agent assigned to the task in top of the delayed task) informs the PA. In our example, that agent is Marteen, responsible for the task Information-Search. Marteen informs the PA of that condition, and provides it with the information required to configure the task with delayed configuration. In the example, the result of P-Search contained a number of items considered insufficient by the capability Search-Assessment. Consequently, Search-Assessment outputs the formula GENERALIZE-QUERY (the reader is referred to §5.7.1 for a more detailed account of this mechanism) to indicate that it is convenient to generalize the scope of the query in order to get more results. Marteen informs the PA that the task M-Search has to be configured using the formula GENERALIZE-QUERY as a new postcondition for the Knowledge Configuration process. Therefore, the PA moves again to the Knowledge Configuration scene to request the K-Broker to configure the task M-Search using the new postcondition.

Figure 7.26 shows a screenshot of the K-Broker internal state during the new Knowledge Configuration process. Notice that the K-Broker has selected the capability Query-generalization to solve the task Adapt-query, since it is the only capability specifying the formula GENERALIZE-QUERY as a postcondition.

After configuring the task Modify-Search, the PA engages a Team Formation scene again to form a team for that task. The agents involved in the original team are waiting in the Teamwork scene, that remains active, but in the meantime, they are requested again to form part of the new team responsible for

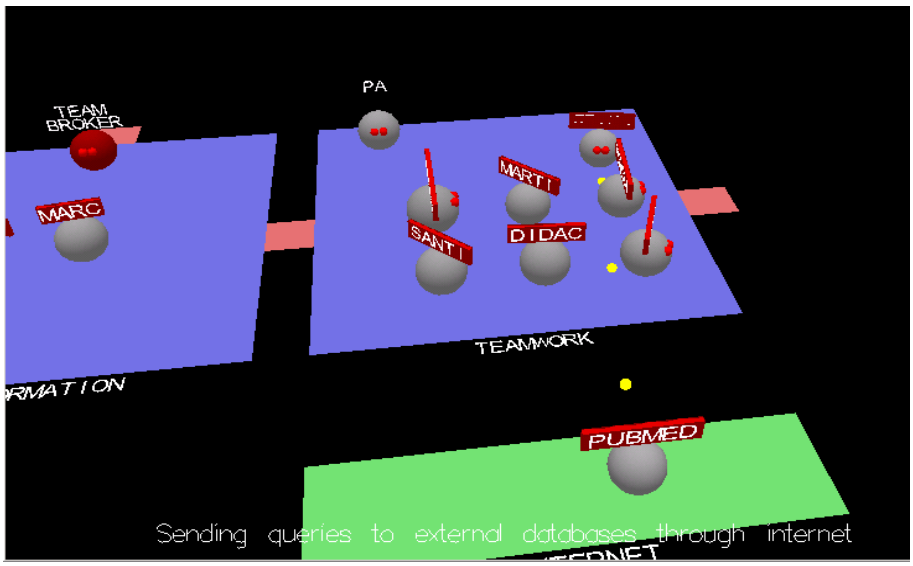


Figure 7.25: Screenshot of the Teamwork scene

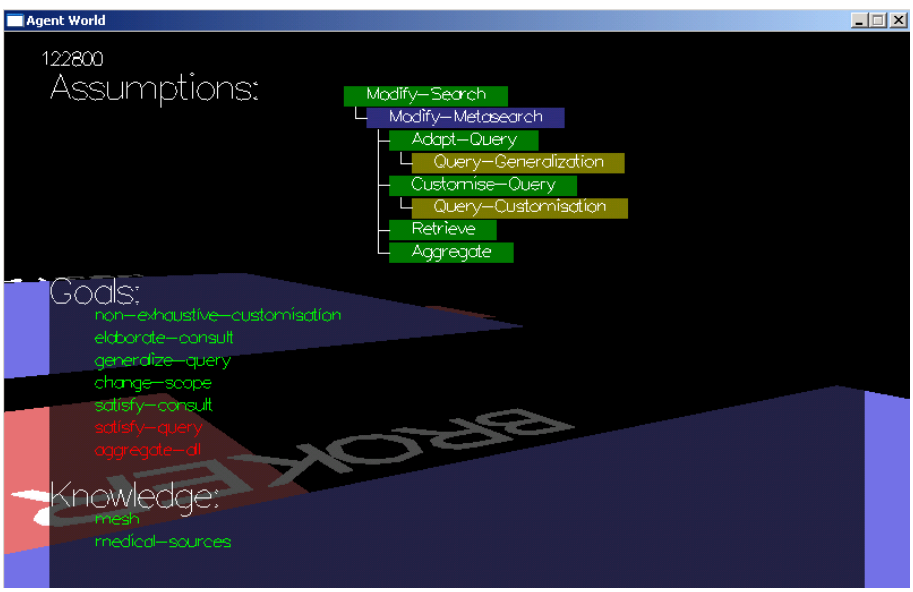


Figure 7.26: Example of the K-Broker internal state for a delayed task

the Modify-Search task. Recall that an agent can participate simultaneously in several scenes, though the Agent World monitoring tool is unable to represent that condition.

Finally, the PA sends the result back to the GUI (or to the agent that originated the CPS process). Figure 7.27 shows the results for the consultation example described above (§7.9.2).

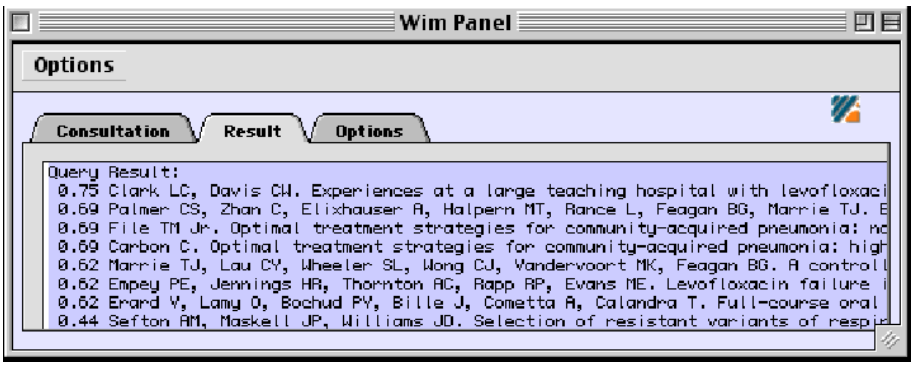


Figure 7.27: Team broker internal state

If the results have to be sent to an external agent using FIPA-ACL, the PA encodes them using XML and sends them within the content of a FIPA inform message to that agent.

```
(inform
    :sender     (agent-identifier :name WIM-PA@\wim\.iiia.csic.es:7778/NOOS)
    :receiver   (set (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE))
    :reply-with     cps-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    cps18236
    :content
       (result
          (action \ldots )
          (Problem-Solution:
             :encoding  <xml? version="1.0" encoding ="ISO-8859-1"?>
          :value
             <scored-items>
                scored-item>
                  <IDENTIFIER>PMID10683053</IDENTIFIER>
                          <RANKING> 0.56</RANKING>
                          <TITLE>A controlled trial of a critical pathway for treatment of
                     community-acquired pneumonia.</TITLE>
                          <AUTHORS>Marrie TJ, Lau CY, Wheeler SL, Wong CJ, Vandervoort MK,
                       Feagan BG.</AUTHORS>
                </scored-item>
                <scored-item>
                  <IDENTIFIER>PMID10418757</IDENTIFIER>
                          <RANKING> 0.34</RANKING>
                          <TITLE>Selection of resistant variants of respiratory pathogens
                     by quinolones.</TITLE>
                          <AUTHORS>Sefton AM, Maskell JP, Williams JD.</AUTHORS>
                </scored-item>
```

```
</scored-items>))))
```

7.10 Other experiments

The WIM application has been developed in the framework of the European Project IBROW, which stands for Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web. The objective of IBROW is to develop intelligent brokers that are able to distributively configure reusable components into knowledge systems through the World-Wide Web, which leads interdisciplinary research on areas such as heterogeneous DB, interoperability and Web technology with knowledge-system technology and ontologies. Concerning the feasibility and the applicability of the IBROW ideas, the project has encouraged the development of interlibrary applications involving components implemented by different partners. Some of these efforts have focused on agent technology, whilst other efforts have turned to new technological developments such as the Semantic Web Services.

Now we are going to review two applications: (1) an application concerning two libraries implemented by agents running on different platforms, and (2) a Semantic Web Service communicating with the WIM application.

7.10.1 Inter-library application

The idea of inter-library applications is to configure applications that involve problem solving components from different libraries, and running in different platforms as well. In such an scenario, three types of interoperation problems arise: semantic, syntactic and relative to the interaction protocols.

At the semantic level, in order to compare (match) specifications from different libraries, either a shared ontology specifying the concepts used by the different libraries or a collection of mappings between the different ontologies is required.

At the syntactic level, a common language, a encoding data format and a serialization mechanism to send and receive data between components from different libraries are necessary, specially when two libraries are implemented over heterogenous platforms. For instance, in the IBROW project we have connected the ORCAS agent platform with a JADE platform through the FIPA-Mediator, using the FIPA-ACL, and either XML or RDF to encode the content of a message.

Finally, agents must share some interaction protocol in order to communicate meaningfully. Therefore, we have developed interaction protocols based in the FIPA Request-Inform protocol and using the FIPA Agent Communication Language.

The interlibrary application involves two libraries: one library is the one provided by the WIM application, running over the NOOS agent platform, which uses LISP; the other library is intended for document classification tasks, has

been developed by the SWI group, from the UVA University, and is implemented over the JADE platform, which uses Java. The are two agents interconnecting both, the FIPA-Mediator running in the NOOS platform, and the NOOS-Proxy running in the SWI agent platform. These (pseudo) agents implement the communication and the data translation between both agent platforms using FIPA-ACL as the communication language, and SL0 as the content language. Figure 7.28 shows the interconnection of both platforms and the agents responsible for the interoperation: a Broker and a User Agent in the SWI platform, and a Librarian and a Personal Assistant in WIM.

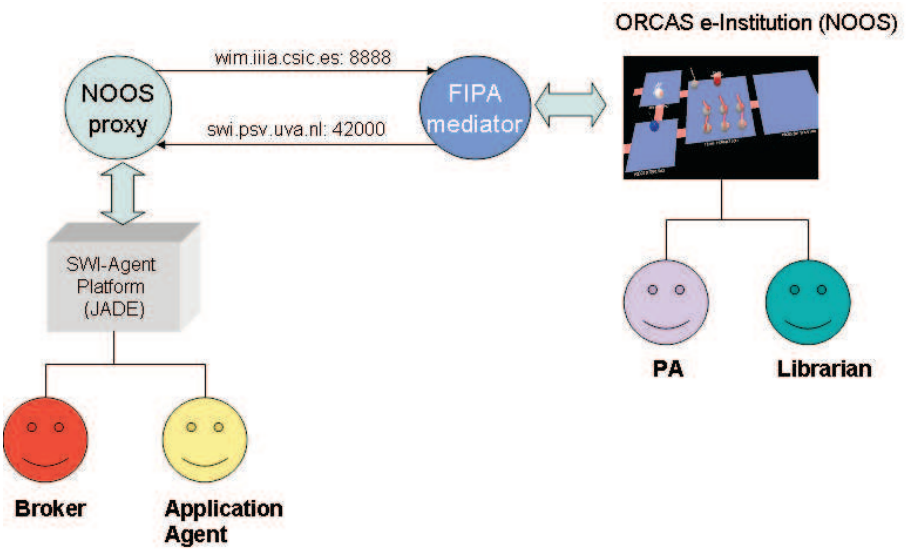


Figure 7.28: Interlibrary application

The SWI Broker asks the WIM Librarian about the tasks and capabilities available in WIM to check whether a configuration of the application task using WIM components is possible. The User Agent interacts with the user to get a specification of the problem and interacts with the SWI Broker to obtain a configuration of components for the application task, that is called Search & Classify. The Search & Classify task is decomposed into three subtasks by the Search, Retrieve & Classify task-decomposer, as shown in Figure 7.29.

The WIM PA acts as a mediator between the SWI broker and the ORCAS brokers: The Knowledge-Broker and the Team-Broker. The idea is that the WIM brokers operate locally over the tasks to be performed by WIM agents, while the global task is configured by the SWI broker, that delegates the configuration and execution of some tasks to WIM agents.

The ORCAS e-Institution in which the WIM application is deployed offers some services to external agents, including the following (Appendix F):

1. *Brokering*: Obtaining a task-configuration according to a specification of

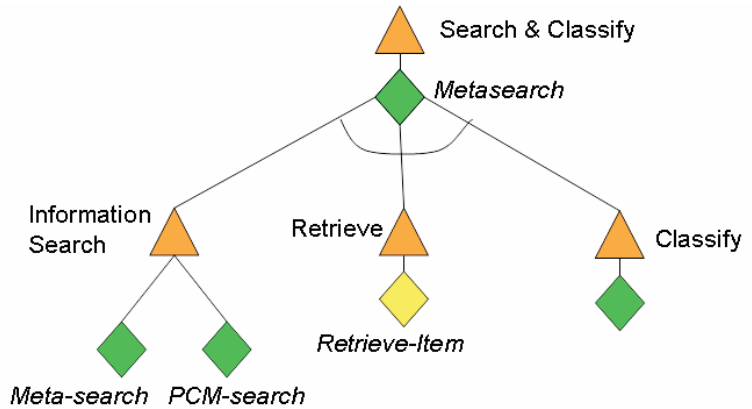


Figure 7.29: Interlibrary application

problem requirements.

2. *Team formation*: forming and instructing a team of agents with the capabilities required by a task-configuration

3. *Teamwork*: performing the teamwork activity to solve a problem instance, given a team-identifier of a previously instructed team.

4. *Cooperative Problem-solving*: this service comprises all the previous services within a single request-inform protocol.

5. *Retrieve specification*: this service provides a pattern-based retrieval service to gather component specifications from the library.

Figure 7.30 shows the subtasks and data involved by the interlibrary application task, Search & Classify. The goal of this task is to search bibliographical references, and then retrieve all extra information available for some references in order to classify them. This task is decomposed into tree subtasks by our task-decomposer: Information-search, Retrieve-extra-info and Classify.

The input data to the Information-search task is a query, and the result is set of items containing bibliographical references, and ranked according to the relevance to the query. These items include only the basic information: an identifier, a title and the list of authors. Next, some of these items are used as inputs of the Retrieve-extra-info task, which is able to retrieve all the information available for a reference, like the language, subject (keywords), abstract, year and so on. Since several items may be selected for classification, then several instances of this task may be required, one for each item selected. Finally, the retrieved items are clustered and classified by some classification method in the SWI library, using the information provided by the WIM task Retrieve-extra-info.

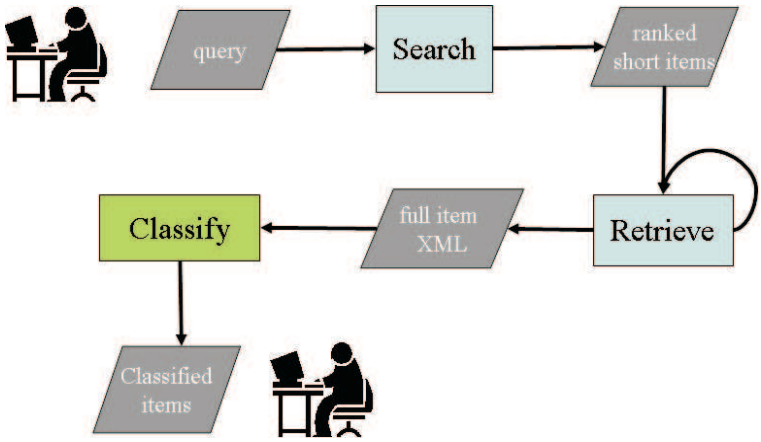


Figure 7.30: Interlibrary application

7.11 Conclusions

Finding relevant information can be described only with respect to the purpose for which this information is sought, for the kind of use we have in mind for that information. However, classical information retrieval (IR) based on keywords assumes all the information provided by the user is flat and in the form of keywords. The documents being searched are abstracted as vectors of keywords to be compared with the "document query" of the user also abstracted as a vector of keywords. This approach focuses on the notions of *content* and *relevance*: the content of documents are expressed as vectors of keywords and the similarity measure that compares them with the query vector expresses the degree of relevance of each document to that query.

Let us consider the query example we have used in this article: a user is interested in information about *guidelines* on the use of *levofloxacin* in the treatment of *pneumonia*. The keyword based approach would recommend simply to use the keywords `(guidelines, levofloxacin, pneumonia)` as a flat representation of the user request. However, we know that the user is in fact talking about a substance (levofloxacin), a disease (pneumonia), and a type of information searched (guidelines).

The approach taken in the Web Information Mediator has been to provide a bridge (a mediation service) between the user's expression of the kind of information to be found and the existing repositories of medical information that use keyword-based retrieval. WIM distinguishes between keywords that define content (e.g. searching documents *about* pneumonia) from categories that define the intended *use* of the information (e.g. using information on *guidelines* instead of, say, *diagnosis*). Categories are thus used to transform the initial user request into a collection of keyword-based queries that include the knowledge needed to find occurrences of keywords related to that category. As we have

seen, different keywords related to *guidelines* are used in these queries—while a keyword-based retrieval would have only used the keyword `guidelines`. The retrieved documents are ranked in accordance to how useful they are for the intended purpose of the category *guidelines*, but they also include documents indexed as *clinical protocols*. Other utility criteria used in WIM are related to specifying a broad and costly search towards a narrow and fast search. We have shown no example here but the idea is very similar in that a thesaurus is used (instead of a category) to elaborate further queries using synonyms and related words. The criterion of utility is here related to the type of search and can be combined with the criteria based on EBM.

Chapter 8

Conclusions and future work

This chapter sums up the contributions of our thesis, compares our contributions with some related works, and presents some open issues and shortcomings to be addressed by future work.

8.1 Introduction

As our work was advancing, we introduced and discussed many ideas, and put some of them in practice, but there are other issues deserving our attention that we could not afford in the context of this thesis. We have been faced with so many difficulties that we have had to focus on a limited set of issues, barely tackling other subject matters. This chapter sums up the outcomes of our efforts and recapitulate both the achievements and the still open issues.

We have stated our aim at the Introduction: to provide a framework for developing and deploying open Multi-Agent Systems supporting the automatic, on-demand configuration of agent teams according to stated problem requirements. As a result of our efforts, we have developed a multi-layered framework for MAS development and deployment that integrates Knowledge Modelling and Cooperative Multi-Agent Systems together. This framework is called ORCAS, which stands for Open, Reusable and Configurable multi-Agent Systems.

The ORCAS framework encompasses three separated layers, the Knowledge Modelling Framework, the Operational Framework, and the Institutional Framework, also called the ORCAS e-Institution. The purpose of separating the framework in layers is to bring developers an extra flexibility in adapting the framework to their own requirements, preferences and needs. Now we are going to review the main features of this framework, focusing on those issues we have made more contributions.

The ORCAS Knowledge Modelling Framework provides a conceptual framework for the design of open MAS that aims at maximizing capability reuse

[Gómez and Plaza, 2004a]. The main features of this framework are, in brief, the following:

- the components of a MAS are described at the knowledge-level [Gómez et al., 2003a], abstracted from any implementation details, and enriched with semantic information specified as shared ontologies;
- there is a clear separation of tasks and capabilities from the application domain, through the specification of domain-models using its own ontologies [Gómez et al., 2001];
- a compositional, bottom up approach to system design is proposed based on two types of matching relations: task-capability matching and capability-domain matching [Gómez and Plaza, 2004a];
- automated, on-demand design of agent teams according to the requirements of the problem at hand (Knowledge Configuration), implemented as a search process supporting three strategies [Gómez and Plaza, 2004b]: interactive, depth first (Search and Subsumes) and best first (Constructive Adaptation); and
- a clear separation of two levels in the description of a MAS: on the one hand, the Abstract Architecture states the type of components used to describe a system, the features characterizing each type of component, and the relations constraining the way components can be connected; on the other hand, the Object Language refers to the representation language used to specify component features and the inference mechanism used to reason about component specifications (to compare components in order to verify a matching relation).

The Operational Framework provides a link between the KMF and cooperative MAS; specifically, it extends the Abstract Architecture to become a full-fledged Agent Capability Description Language (ACDL), and describes an alternative model of the Cooperative Problem Solving (CPS) process based on a Knowledge Modelling approach [Gómez and Plaza, 2004b]. The main features of the ORCAS ACDL addressed by the Operational Framework are the following:

- the ORCAS ACDL includes both the communication aspects required to enact an agent capability, and the operational description of a task-decomposer, specifying the control flow among subtasks;
- the communication and the operational description of capabilities are decoupled from the functional aspects, thus maximizing the reuse of agent capabilities through different interaction protocols; and
- both the communication and the operational description of a capability are specified at the macro (social) level, using concepts from the electronic institutions formalism.

The main characteristics of the ORCAS model of the CPS process are the following:

- it includes a Problem Specification stage and a Knowledge Configuration stage to guide the Team Formation process according to the requirements of the problem;
- the different stages of the CPS process can be interleaved, allowing an agent society to react to runtime events and adapt to the dynamic nature of open environments; and
- the multiagent planning stage is partially substituted by the Knowledge Configuration process, and somehow by the team configuration stage carried out during the Team Formation process.

The Institutional Framework describes an implemented infrastructure for developing and deploying Multi-Agent Systems following the electronic institutions formalism. The ORCAS framework has been implemented as an electronic institution [Esteva et al., 2002b] in which institutional agents are responsible for mediating between providers and requesters of problem-solving capabilities [Gómez et al., 2003b].

While the KMF and CPS processes provide the tools to design the competence and the social requirements of agents, the ORCAS e-Institution provides the protocols for registering services, configuring tasks and customizing agent teams to solve those tasks. It is a tool to deploy flexible, extensible and configurable Multi-Agent Systems. An application of this infrastructure has been successfully developed and tested, WIM, a configurable meta-search application [Gómez and Abasolo, 2002] in a medical domain [Gómez et al., 2002]. The ORCAS e-Institution has set a precedent as the first multiagent infrastructure based on the electronic institutions approach. Moreover, we are somehow contributing to the field of electronic institutions by providing a framework for configuring electronic institutions dynamically, out of scenes and performative structures describing the communication and operational aspects of agent capabilities.

We have summarized the main features of the ORCAS framework, and now we are going to discuss some of this features, comparing them with related work.

8.2 Discussion

With the introduction of the knowledge level [Newell, 1982] in the development of Knowledge Based Systems, the knowledge acquisition phase turned from a knowledge transfer approach to a model construction approach [Clancey, 1989, Studer et al., 1998]. Knowledge Modelling Frameworks propose methodologies, architectures and languages for analyzing, describing and developing knowledge systems [Steels, 1990, Chandrasekaran, 1986, McDermott, 1988, Schreiber et al., 1994a, Fensel et al., 1999]. The goal of a KMF is to provide a conceptual model of a system that describes the required knowledge and inferences at an implementation independent way. This

model is intended to support the engineer in the knowledge acquisition phase [Van de Velde, 1993] and to facilitate reuse [Fensel, 1997a]. The reuse issue has received a lot of attention last years from the knowledge modelling community [Benjamins et al., 1996a, Motta, 1999, Fensel and Motta, 2001]; focusing on ontology-based reuse [John H. Gennari and Musen, 1998, Studer et al., 1996] and automated reuse mechanisms [Gaspari et al., 1999, Motta et al., 1999, Fensel and Benjamins, 1998b].

Surprisingly, Knowledge Modelling Frameworks have been rarely applied in the field of MAS to deal with the reuse and interoperation problems; two exceptions are found in [Iglesias et al., 1997, Glaser, 1996], which have adapted the CommonKADs methodology [Schreiber et al., 1994a] to provide a MAS development methodology. Unlike the former proposals, instead of providing a methodology, we provide a conceptual framework that aims to maximize the reuse of agent capabilities across different application domains. Furthermore, we have developed a MAS infrastructure supporting the on-demand configuration of agent teams according to the requirements of the problem at hand. An outstanding difference between our framework and other frameworks is the inclusion of domain-models as an independent entity, and the definition of a matching relation between domain-models and capabilities, so as to facilitate reuse of agent capabilities across different domains.

Herein, we adhere to the view of Internet as an open environment where providers and requesters of capabilities meet and interact to solve specific problems by using the resources at hand. This view of Internet as a distributed computational platform is in spirit the same of the Semantic Web initiative, in particular, our view of agent capabilities shows some aspects closer to issues from the Semantic Web. From the Semantic Web approach building an application is basically a process of composing, connecting and verifying the properties of Semantic Web Services (SWS) in a way that resembles our compositional approach to team design.

There are, however, two outstanding differences between the Semantic Web and ORCAS. On the one hand, ORCAS agents are autonomous entities that can decide to accept or to refuse a request, while services are reactive, passive entities which are directly invoked by the client; therefore, instead of a centralized composition of services, we view the composition of capabilities as a negotiation process among autonomous agents. On the other hand, our language for describing capabilities is domain independent, thus it is intended to maximize reuse, while Web Services frameworks ignore this issue, since they are domain dependent by nature (a Web service is associated to some concrete domain, like the weather of a specific country in a weather forecasting service).

8.2.1 On Agent Capability Description Languages

Some of the first languages for describing agents in open environments were based on logical deduction languages like Prolog; two well known examples are the Interface Communication Language (ICL) used in the Open Agent Architecture [Martin et al., 1999, Cheyer and Martin, 2001] to describe agents as goals, and

LDL++, used in the InfoSleuth infrastructure [Nodine et al., 1999]. Nonetheless, our way of describing the components of a Multi-Agent System is more similar to LARKS [Sycara et al., 2002], a newer language used in the RETSINA infrastructure [Sycara et al., 2001] for describing agent capabilities and performing matchmaking.

The major difference between the languages above and ORCAS lies up in the ORCAS KMF, which decouples the specification of tasks and capabilities from the application domain in order to maximize reuse. Moreover, while RETSINA relies on HTN (Hierarchical Task Network) planning and scheduling, the ORCAS framework substitutes plans by task-configurations, and keeps the scheduling activities out of the framework due to its endodeictic nature (belonging to the agent architecture and thus falling into the micro-level, whereas we are focused on the macro, social level). Moreover, while existing frameworks for MAS cooperation usually assume that plans are obtained beforehand (prior to engage in cooperative activities) or provided by the user, our proposal is to obtain the task-configuration on-demand, out of the capabilities and knowledge available at the moment of receiving a request.

8.2.2 On MAS Coordination and Cooperation

Despite the large research efforts done in the field of Cooperative Problem Solving (CPS), most of the work done falls into one or several stages of the CPS process as presented in [Wooldridge and Jennings, 1994], which has four stages: recognition, team formation, planning and execution. The problem solving process starts with an agent willing to solve a task and realizing the potential for cooperation. The process until the task to be solved is decided is usually skipped, assuming that it is already given [Wooldridge and Jennings, 1999]. Moreover, task allocation among cooperating agents requires some kind of preplan specifying how to decompose a task into subtasks [Shehory and Kraus, 1998], without specifying the algorithms to build such plan, neither the criteria to be taken into account.

In this thesis we have studied the feasibility and utility of a componential, bottom up design approach to build something similar to an initial plan, what we call a task-configuration. When addressing the problem of configuring a team according to problem requirements we agree with other researchers that users matter [Erickson, 1996b]; people may need to understand what happened and why a system alters it response, to have some control over the actions of a system, even when agents are still autonomous, and furthermore, users may need to predict what will happen.

Some frameworks have addressed the question of the user; we can mention for instance the Guided Team Selection approach described in [Tidhar et al., 1996], the top-down search approach proposed in [Clement and Durfee, 1999], and the case-based conversational broker described in [Munoz-Avila et al., 1999].

The main difference of our approach is that a task-configuration contains more information than a Hierarchical Task Network, since it includes domain-models in addition to tasks and capabilities (equivalent to actions in an HTN).

We claim that our separation of the knowledge and the operational aspects involved in the CPS process helps understand the aspects underpinning agent cooperation, since it accounts for the static vs dynamic dimension of agent societies. The idea is to exploit the fact that the specification of agent capabilities remains stable for long periods of time, whereas there are dynamic aspects of the system or its environment that change very quickly, i.e. the agent workload or the network traffic. Therefore, it is useful to make a configuration in terms of the static description of capabilities, and thereafter the static configuration can be used to select the "best" candidate agents[1] according to dynamic and context-based information.

While other frameworks and infrastructures concentrate on the task allocation stage carried on during Team Formation, the ORCAS Knowledge Configuration process is situated before Team Formation in the Cooperative Problem-Solving model. This does not mean, however, that the Knowledge Configuration process should be completed prior to initiate Team Formation; in fact, we have proposed strategies to interleave both activities with the execution: distributed configuration, lazy configuration, and dynamic reconfiguration.

8.2.3 On Semantic Web Services

From the Semantic Web approach Internet is viewed as a network of distributed and heterogeneous services that must be composed and "orchestrated" to achieve complex tasks. From that view, a Service Description Language must support just the same type of activities we want to be supported by the ORCAS ACDL: discovery, execution, composition and interoperation. Actually, there is ongoing work to put services and agents together, for instance, the Web Service Modelling Framework (WSMF) [Fensel and Bussler, 2002] and the DAML-S ontology [The DAML-S Consortium, 2001] are being developed with a similar set of requirements in mind, though the same concepts are expressed using a different vocabulary.

Our way of describing the functional aspects of a capability is equivalent to a service profile in DAML-S, the communication supported by an agent over a capability corresponds to the grounding of a service, and the operational description of a task-decomposer plays a role similar to the process model of a service.

There are minor technical differences concerning the communication aspects of the two approaches. There is, however, an outstanding conceptual difference: agents are autonomous entities capable of refusing a request, whereas Web Services are passive entities that are directly invoked by the requester. Consequently, agents engage in cooperation following some kind of negotiation or communication activity in which they take an active role, whereas Web Services are composed and "orchestrated " under the baton of another entity (usually through the use of workflow-based specifications) without active participation of

[1]The notion of agent goodness is specified as a criteria to be optimized, i.e. cost, speed, reliability and so on.

DAML-S	Agent activities	ORCAS ACDL
Profile	Discovering (matchmaking)	Inputs, outputs and competence
Grounding	Invocation and Execution	Communication
Operational model	Composition and Interoperation	Subtasks and operational description

Table 8.1: Comparison of the ORCAS ACDL against DAML-S

the Web Services.

Table 8.1 summarizes the relation between the features characterizing a capability in ORCAS, the features proposed to describe agent-enabled Semantic Web Services in the DAML-S ontology [The DAML-S Consortium, 2001], and the kind of activities these features are required for [Bansal and Vidal, 2003, Bryson et al., 2002, Park et al., 1998, Payne et al., 2001].

8.2.4 On the design of agent teams

Design is a fundamental aspect of engineering in general and software development in particular, and there are some efforts to provide design methodologies for agents and even multiagent systems, but the idea of design has not been applied to coalition or team formation. Therefore, our idea of introducing a design perspective in the team formation process is new. Specifically, we have introduced a design stage, that we called the Knowledge Configuration process. The Knowledge Configuration process aimsat deciding the competence required for a team of agents to achieve a goal satisfying global requirements. This process has been implemented as a search process over the space of possible task configurations (designs). Such a search-based approach opens the door to a large number of techniques to be applied, and CBR is just an example of that (§4.5).

8.3 Future work

The objectives of this work were ambitious; the complexity of the problems faced and the wide, interdisciplinary scope of the challenges encountered make us concentrate on some specific aspects, while others issues have been disregarded or postponed. This section will introduce still open problems and will draw up some research lines to be followed by future work in order to advance forwards towards the realization of full-fledged open MAS.

Open systems allow the involvement of agents from diverse design teams, with diverse objectives that may be unknown at the time of design. Multiagent infrastructures are expected to provide a critical enabler for development of scalable interoperable systems, however, in order to successfully communicate in such an environment, agents need to overcome two fundamental problems: first, they must be able to *find* each other (since agents might appear or disappear at any time), and once they have done that, they must be able to *interact*

[Nwana and Woolridge, 1996, Jennings et al., 1998]. Although existing infrastructures are incorporating mechanisms for advertising, finding, using, combining and updating agent services and information [Decker et al., 1997b], most of them are still relying on homogeneity assumptions to achieve a successful interaction: a common communication language and protocol; a common format for the content of communication; and a shared ontology.

As stated in the Introduction (Chapter 1), full-fledged open MAS must support cooperative work spanning multiple application domains and assembly of teams out of heterogeneous agents (and legacy applications) developed by different teams, using different communication languages and ontologies. For this reason, there is a standing interest on semantic interoperability and semantic middleware.

In the ORCAS KMF, components can be described using its own, independent ontologies [Fensel et al., 1997]. Because of this conceptual decoupling, ontology mappings may be required to match capabilities to tasks and domain-models to capabilities when there is an ontology mismatch between two specifications. Nevertheless, we have focused here on the matching relations, assuming that the necessary ontology mappings are already built, or assuming that all the components share the same ontologies. This is a reasonable assumption, since it is feasible and convenient to built the mappings beforehand, previously to make a component available for its use. But we expect ontology engineering to become a very important ingredient of agent middleware, as well as other fields in which semantic interoperability may play a role (e.g. Semantic Web Services and Cooperative Information Systems).

Taking into account our general motivations, some lines of research deserving further attention are those concerned with ontology engineering, including the following topics:

- ontology alignment and mapping;
- languages for representing mappings; and
- metrics for measuring semantic similarity and semantic distance.

The ORCAS framework could highly benefit from technological advances supporting the automatization of ontology-related activities like reasoning with mappings and mapping discovery. The idea of introducing a new kind of reusable *connectors* to bridge the gap between semantically differing specifications seems considerably interesting. Some inspiring works concerning that subject are found in [Park et al., 1998, Gómez and Benjamins, 1999]. In ORCAS connectors could be inserted between capabilities and domain-models, and between tasks and capabilities as well. A connector in ORCAS involves two dimensions: a knowledge-level specification, which allows to match components specified with different ontologies; and the implemented counterpart, which allows semantically (or syntactically) heterogeneous agents to interoperate during the Teamwork process.

The idea of ontology agents specialized in discovering mappings fits well in the context of the ORCAS infrastructure, since ORCAS relies on institutional

agents providing specialized services to other agents (e.g. the Librarian and the Team-Broker).

Other aspects of our work deserving a deeper study are those concerning the adaptation of agent teams to the dynamic nature of open environments to deal with unexpected events and handle errors. We have already draw the notions of reconfiguration, delayed configuration and lazy configuration as extensions to the core model of the ORCAS Operational Framework. We think a lot of work remains to be done yet in order to improve the adaptability of agent teams to changes in the environment, like introducing learning or incorporating some kind of meta-strategy to decide the better strategy at each particular moment.

Another point that can be addressed by future work is the extension of the ORCAS ACDL to provide fully declarative description for the operational model of a task-decomposer, including not only control flow but also intermediate data processing. Such a feature will allow task-decomposers to exist in a purely declarative form, thus any agent understanding that language would be able to follow a task-decomposer, rather than having specific agents implementing each new task-decomposer.

The ORCAS e-Institution is subject to large modifications and improvements; for instance, we think two interesting areas to work upon are security matters and extended interoperation.

Concerning security, we have not considered any security issues yet in the ORCAS framework, though we are aware of their critical importance to the field in order to develop industrial and commercial applications. An interesting line to be followed through is that of introducing specialized agents to take care of supervision and security tasks, like sentinels [Dellarocas and Klein, 1999] and governors [Esteva et al., 2002b].

Concerning a greater support to interoperation, the idea of federated electronic institutions raises as a very interesting concept. The point is to allow configuring agent teams out of agents running in separate agent infrastructures. Although we have carried out some experiments involving several libraries and heterogenous agent platforms (we have connected the NOOS Agent Platform, that uses Lisp, and JADE, that uses Java), they have been conducted on a rather ad-hoc manner; therefore, we think the ORCAS framework could be improved by including a principled approach to form federations embracing several ORCAS compliant infrastructures.

Still another issue deserving work in the future is the inclusion of contractual mechanisms to support e-Commerce applications like supply chains, auctions and e-markets. Concerning this research line, we are thinking about the aspects required to implement what we like to call "terms of commitment", a mechanism to agree upon by team members when accepting a team-role. We envision agent societies negotiating the terms of service to which agents commit to partake in a team. An ontology of possible terms of commitment (e.g. exclusive vs. non exclusive, pulls vs. push, quality of service measures, cost, etc.) together with some interaction protocols has to be developed to deal with such a kind of negotiation.

Finally, we want to mention the advisability of a complete methodology for the design an development of MAS according to the ORCAS framework.

Appendix A

Specification of the Knowledge Modelling Ontology

```
(in-package noos)

(define-ontology KM-Ontology
  (creator "IIIA - CSIC")
  (description "KM-Ontology describes the elements of the \orcas\ KMF"))

(define-sort KM-Ontology)

(define-sort (KM-Ontology Concept))

(define-sort (KM-Ontology Binary-Relation)
  (argument1 Concept)
  (argument2 Concept))

(define-sort (Concept Knowledge-Component)
  (name String)
  (pragmatics Pragmatics)
  (ontologies Ontology Empty-Set))

(define-sort (Knowledge-Component Ontology))

(define-sort (Knowledge-Component Task)
  (uses Task Empty-Set)
  (input-roles Var Empty-Set)
  (output-roles Var Empty-Set)
  (competence Competence)
  (assumptions Formula))
```

```
(define-sort (Knowledge-Component Domain-Model)
  (uses Domain-Model Empty-Set)
  (properties Formula Empty-Set)
  (metaknowledge Formula Empty-Set)
  (knowledge Signature-Element Empty-Set))

(define-sort (Knowledge-Component Capability)
  (communication Communication)
  (input-roles Var Empty-Set)
  (output-roles Var Empty-Set)
  (competence Competence))

(define-sort (Capability Task-Decomposer)
  (subtasks Task Empty-Set)
  (operational-description Operational-Description))

(define-sort (Capability Skill)
  (knowledge-roles Signature-Element Empty-Set)
  (assumptions Formula Empty-Set))

(define-sort (Concept Pragmatics)
  (title string)
  (creator string)
  (subject string)
  (description string)
  (publisher string)
  (other-contributor string)
  (date string)
  (resource-type string)
  (format string)
  (resource-identifier string)
  (source string)
  (language string)
  (relation string)
  (rights-mangement string)
  (last-date string)
  (be-used string)
  (evaluation string)
  (application-descriptors Pragmatics-Descriptor Empty-Set))

(define-sort (Concept Pragmatics-Descriptor)
  (name string)
  (value string))

(define-sort (Concept Competence)
  (preconditions Formula)
  (postconditions Formula))

(define-sort (Concept Signature-Element)
```

```
  (name String))

(define-sort (Concept Formula)
  (name String))

(define-sort (Concept Operational-Description)
  (intermediate-roles Signature-Element Empty-Set)
  (programs string))

(define-sort (Concept Renaming)
  (in Signature-Element)
  (out Signature-Element))

(define-sort (Concept Communication)
  (communication string))

(define-sort (Binary-Relation Adapter)
  (argument1 Knowledge-Component)
  (argument2 Knowledge-Component)
  (pragmatics Pragmatics)
  (ontologies Application-Ontology Empty-Set)
  (renamings Renaming Empty-Set))

(define-sort (Adapter Bridge)
  (uses Bridge Empty-Set)
  (mapping-axioms Formula Empty-Set)
  (assumptions Formula Empty-Set))

(define-sort (Bridge Capability-Domain-Bridge)
  (argument1 Capability)
  (argument2 Domain-Model)
  (uses Capability-Domain-Bridge Empty-Set))

(define-sort (Bridge Capability-Task-Bridge)
  (argument1 Capability)
  (argument2 Task)
  (uses Capability-Task-Bridge Empty-Set))

(define-sort (Bridge Task-Domain-Bridge)
  (argument1 Task)
  (argument2 Domain-Model)
  (uses Task-Domain-Bridge Empty-Set))

(define-sort (Adapter Refiner)
  (in Knowledge-Component)
  (out Knowledge-Component))

(define-sort (Refiner Ontology-Refiner)
  (in Application-Ontology)
  (out Application-Ontology))
```

```
(define-sort (Refiner Domain-Refiner)
  (in Domain-Model)
  (out Domain-Model)
  (properties Formula Empty-Set)
  (metaknowledge Formula Empty-Set)
  (knowledge Formula Empty-Set))

(define-Sort (Refiner Task-Refiner)
  (in Task)
  (out Task)
  (input Input-Roles)
  (output Output-Roles)
  (competence Competence)
  (assumptions Formula Empty-Set))

(define-sort (Refiner Capability-Refiner)
  (in Capability)
  (out Capability)
  (communication Communication)
  (input Input-Roles)
  (output Output-Roles)
  (competence Competence))

(define-sort (Capability-Refiner Task-Decomposer-Refiner)
  (in Task-Decomposer)
  (out Task-Decomposer)
  (subtasks Task Empty-Set))

(define-sort (Capability-Refiner Skill-Refiner)
  (in Skill)
  (out Skill)
  (knowledge Signature-Element Empty-Set)
  (assumptions Formula Empty-Set))
```

Appendix B

Formalization of the Query Weighting Metasearch Approach

The capabilities used in the WIM application are based on a query weighting framework [Gómez and Abasolo, 2003] that is applied to transform queries during the query adaptation stage: to transform the user query into a collection of domain queries, and to transform each domain query into a collection of source queries. This framework relies on a keyword based representation of queries, plus the use of search filters, which are the common elements used by existing search engines in the Web.

A *query* Q is defined as a vector of non-repeated elements, which can be keywords, search filters or another element.

$$Q = \langle k_1 \dots k_n \rangle \qquad \forall i, j : 1 \leq i, j \leq n; k_i \neq k_j \tag{B.1}$$

A *weighted-query* QW is a pair composed of a query and a weight in the interval [0, 1].

$$WQ = \langle Q, w \rangle \qquad w \in [0, 1] \tag{B.2}$$

A *query-transformation* τ is a relation between two queries (Q_1, Q_2) and a weight (w), defined as follows:

$$\tau(Q_1, Q_2, w) \Leftrightarrow (\exists! k | k \in Q_1 \wedge k \notin Q_2) \wedge (\exists! k' | k' \in Q_2 \wedge k' \notin Q_1) \wedge \sigma(k, k', w) \tag{B.3}$$

where k and k' are elements of queries, and $\sigma(k, k', w)$ is a relation between two elements and a weight in the interval [0,1].

Intuitively, this definition means that one query is exactly like the other but a single query element. In other words, there is a unique element k in Q_1 not in Q_2,

and there exists an unique element k' in Q_2 not in Q_1, such that these elements are related by a weight $(\sigma(k, k', w))$. If there exists a query transformation between two queries and a weight w, one can transform one query into the other by replacing the element k by k', assigning the new query with a weight $= w$.

The query transformation relation is used to weight a query during a query adaptation process. In WIM, domain queries are generated from the original user consultation by using domain knowledge, such as a thesaurus or a collection of knowledge categories. The idea is that if one query is the result of a query transformation, then we can weight the new query with the weight relating both the original and the new query.

Notice that the weighted relationships between query elements (i.e. between two keywords) are encoded or can be derived from application domain knowledge. For instance, WIM uses a thesaurus to obtain semantic relationships between keywords, applying a mapping from the qualitative relationships (e.g. synonym) defined in the thesaurus to numeric values which are then used as weights (e.g. two synonyms are related with a weight $w = 1$). Moreover, during the query customization stage, domain queries are transformed into source queries by using a description of information sources. A description of a information source contains a mapping from concepts specified at the domain level to concepts used by a particular information source. The weight applied to a source query resulting of transforming a domain query into a source query (query customization) is obtained from the description of that source. The weighting values depend on the relation between the domain level concept and the source level concept, and have been decided during the knowledge acquisition phase with the help of an expert in medical bibliography.

We have not discussed yet how to assign weights when two or more query elements are changed between two queries, or how to assign a weigh to an already weighted query. Both problems are in fact the same, how to combine or synthesize weights.

Different functions can be used to combine weights. In the Query Weighting framework one can consider weights as membership values with respect to the user interest when posing a query, as well as logical values expressing the degree of relevance or utility of a query with respect to the user query. Therefore, weights can be combined by using numerical aggregation operators or multivalued logical operators (e.g. t-norms). The Query Weighting framework states a general rule that constrains the type of query synthesizing functions allowed. This rule states that the *weight of a query cannot be increased after applying a transformation*; the meaning is that query transformations move queries further away from the user request. In other words, if we transform a query q with a weight w into a new query q' with a weight w', w' cannot be greater than w. Such class of operators includes -but is not reduced to- the family of t-norm operators.

The composition or synthesis of weights is defined from the notion of a *chain of query transformation*. A chain of query transformations T between two queries indicates that there exist a sequence of query-transformations between the two

queries. A *chain of query transformations* T is defined as a relation between two queries and a weight, as follows:

$$T(Q_1, Q_2, w) \Leftrightarrow \begin{cases} (Q_1, Q_2, w) \wedge \\ \exists Q', \tau | \tau \wedge \tau(Q', Q_2, w') \wedge T(Q_1, Q', w'') \\ \wedge w = \Theta(w', w'') \end{cases} \tag{B.4}$$

where Θ is a *t-norm* operator.

Now we are going to define the functions used to obtain the weight for a query after applying a query transformation.

A *Query-weighting* function Γ is a function to obtain a weight between two queries according to the *chain of query-transformations* between those queries. This function is defined as follows:

$$\Gamma : \mathbb{Q} \times \mathbb{Q} \rightarrow [0, 1]$$

$$\Gamma(Q_1, Q_2) = \begin{cases} w & iff \quad T(Q_1, Q_2, w) \\ 0 & otherwise \end{cases} \tag{B.5}$$

where Q_1, $Q_2 \in \mathbb{Q}$, are queries and T is a *chain of query-transformations*.

A *weighted-query-weighting* function Ω is a function to calculate a weight according to the *chain of query-transformations* between a weighted query and a non-weighted query. Given a query Q_1 and a weighted query $\langle Q_2, w \rangle$, we define a *weighted-query-weighting* as follows:

$$\Omega : \mathbb{Q} \times \mathbb{Q} \times \mathbb{W} \rightarrow [0, 1]$$

$$\Omega(Q_1, Q_2, w) = \begin{cases} \Theta(w, w') & iff \quad T(Q_1, Q_2, w') \\ 0 & otherwise \end{cases} \tag{B.6}$$

where Q_1, Q_2, are queries, w is the weight assigned to one of the queries, T is a *chain of query-transformations* and Θ is a *t-norm* operator.

The *query-weighting* and *weighted-query-weighting* functions are used to obtain the weight for the query resulting of applying one or more query transformations. The former is used when the original query is not weighted, and the last when the original query is already weighted. In fact, both functions can be reduced to a unique function if we consider the non-weighted queries as having a weight equal to 1.

When query is weighted after applying a transformation, this expresses the relative importance or representativity of that query with respect to the original one. The meaning of a weight assigned to a query is logically inherited by the documents or items retrieved for that query, thus we can say that the weights associated to the items retrieved represent the membership of those elements to the topic requested by the user.

Appendix C

Specification of the ORCAS e-Institution

```
(in-package noos)

(define-institution \orcas\_institution as
  dialogic-framework = \orcas\-dialogical-framework
  performative-structure = \orcas\_performative_structure
  norms = ())

(define-performative-structure \orcas\_performative_structure as
    scenes = (
        (Root Root-Scene)
        (Output Output-Scene)
        (Brokering Brokering-Scene)
        (Registering Registering-Scene)
        (Team-formation Team-Formation-Scene)
        (Problem-solving Problem-Solving-Scene)
        (Request-inform Request-Inform-Scene)
        (Request-wrapper Request-Wrapper-Scene)
        (Full-Problem-solving Full-Problem-Solving-Scene))
    transitions = ((T0 AND-AND)
                (T1 AND-AND)
                        (T2 AND-AND)
                (T3 AND-AND)
                        (T4 AND-AND)
                        (T5 AND-AND)
                (T6 AND-AND)
                        (T7 AND-AND))
    connections = (
        (Root T0((x PSA)(y Librarian)))
                 (Root T1((x PA)(y Broker)))
                 (Root T2 ((x Team-broker)))
```

```
                (Registering T1((z Librarian)))
        (Registering T2((x PSA)))
        (Brokering T2((x PA)))
                (Brokering T3((x Librarian)))
        (Team-formation T5((y Team-broker)))
                (Team-formation T4((x PA)(z PSA)))
        (Problem-solving T6 ((x PA) (y PSA)))
        (T0 Registering((x PSA)))
        (T0 Registering((y Librarian)))
        (T1 Brokering((x PA)))
                (T1 Brokering((y Broker)))
                (T1 Brokering((z Librarian)))
        (T3 Output((x Librarian)))
                (T3 Output((y Broker)))
                (T2 Team-formation((z PSA)))
                (T2 Team-formation((y Team-broker)))
                (T2 Team-formation((x PA)))
                (T5 Output((x Team-broker)))
                (T6 Output((x PA)))
        (T6 Output((y PSA)))
                (T4 Problem-solving((x PA)))
                (T4 Problem-solving((y PSA)))
                (Root T7 ((x Requester) (y PA)))
                (T7 Full-Problem-solving ((x Requester) (y PA))))
    initial-scene = Root
    final-scene = Output)

(define-scene Root-Scene as
    roles = (PSA PA Librarian Broker)
    scene-dialogic-framework = IBROW-Library
    states = (W0)
    initial-state = W0
    final-states = (W0)
    acces-states = ((PSA (W0)) (PA (W0)) (Librarian (W0)) (Broker (W0)) )
    exit-states = ((PSA (W0)) (PA (W0)) (Librarian (W0)) (Broker (W0)) )
    connections = (
    ))

(define-noos-scene Team-Formation-Scene
  :description "The Team Broker forms a team to solve a problem"
  :roles  (PA PSA Team-broker)
  :states  (W6 W5 W4 W3 W2 W1 W0)
  :initial-state  W0
  :final-states  (W6)
  :connections  ((W0 W1 (request (?x PA) (?y Team-broker) Task-Configuration))
                 (W1 W2 (inform (!y Team-broker) (All PSA) Start-Team-formation))
                 (W2 W3 (request (!y Team-broker) (All PSA) Team-role))
        (W3 W3 (accept (?z PSA) (!y Team-broker) Team-role))
                 (W3 W3 (refuse (?z PSA) (!y Team-broker) Team-role))
                 (W3 W2 (inform (!y Team-broker) (?z PSA) Team-role))
```

```
                (W2 W4 (inform (!y Team-broker) (All PSA) Start-Team-configuration))
        (W4 W4 (commit (!y Team-broker) (?z PSA) Team-role))
        (W4 W5 (inform (!y Team-broker) (All PSA) Finish-Team-configuration))
                (W5 W6 (inform (!y Team-broker) (!x PA) Finish-Team-formation))
                (W2 W5 (inform (!y Team-broker) (All PSA) Finish-Team-configuration))
                ))


(define-noos-scene Registering-Scene
  :description "PSAs register their capabilities to the Librarian"
  :roles  (PSA Librarian)
  :states  (W2 W1 W0)
  :initial-state  W0
  :final-states  (W2)
  :connections  ((W0 W1 (Register (?x PSA) (?y Librarian) Capability-Set))
                (W1 W2 (Inform (!y Librarian) (!x PSA)  Capability-Set))
                ))



(define-noos-scene Brokering-Scene
  :description "PA request Broker for a Constructive Adaptation or First-Depth
              Search configuration"
  :roles  (Librarian Broker PA)
  :states  (W6 W5 W4 W3 W2 W1 W0)
  :initial-state  W0
  :final-states  (W6)
  :connections  ((W0 W1 (request (?x PA) (?y Broker) Problem-Specification))
                (W1 W2 (request (!y Broker) (?z Librarian) any))
                (W2 W3 (inform (!z Librarian) (!y Broker) Library))
                (W3 W4 (inform (!y Broker) (!x PA) Broker-Message))
                (W4 W5 (request (!x PA) (!y Broker) GUI-Message))
                (W4 W6 (accept (!x PA) (!y Broker) Broker-Message))
                (W5 W6 (inform (!y Broker) (!x PA) Broker-Message)))
                )

(define-noos-scene Problem-Solving-Scene
  :description "PA request a team of PSAs to solve a problem"
  :roles  (PSA PA)
  :states  (W2 W1 W0)
  :initial-state  W0
  :final-states  (W2)
  :connections  ((W0 W1 (Request (?x PA) (?y PSA) Start-Problem-solving))
                (W1 W2 (Inform (!y PSA) (!x PA)  Finish-Problem-solving))))

(define-scene Output-Scene as
    roles = (Broker Librarian PA PSA)
    states = (W0)
    initial-state = W0
    final-states = (W0)
```

```
    acces-states = ((PA (W0)) (PSA (W0)) )
    exit-states = ((PA (W0)) (PSA (W0)) )
    connections = ())

(define-noos-scene Request-Inform-Scene
  :description "PSA requests another PSA in the team to solve some subtask"
  :roles  (PSA)
  :states  (W2 W1 W0)
  :initial-state  W0
  :final-states  (W2)
  :connections  ((W0 W1 (Request (?x Requester) (?y Informer) Start-Problem-solving))
                 (W1 W2 (Inform (!y Informer) (!x Requester)  Finish-Problem-solving))))


(define-noos-scene Request-Wrapper-Scene
  :description "PSA request a Wrapper to query some information source"
  :roles  (PSA)
  :states  (W2 W1 W0)
  :initial-state  W0
  :final-states  (W2)
  :connections  ((W0 W1 (Request (?x Requester) (?y Informer) any))
                 (W1 W2 (Inform (!y Informer) (!x Requester)  any))))

(define-noos-scene Full-Problem-Solving-Scene
  :description "External agent or GUI request PA to solve a problem using all
                the steps (Full mode)"
  :grid ((3 . 1)
         ((W0 0 0) (W1 1 0) (W2 2 0))
         ((W0 W1 :U) (W1 W2 :U)))
  :roles  (PA Requester)
  :states  (W2 W1 W0)
  :initial-state  W0
  :final-states  (W2)
  :connections  ((W0 W1 (Request (?x Requester) (?y PA) Full-Problem))
                 (W1 W2 (Inform (!y PA) (!x Requester)  any))))
```

Appendix D

Specification of the ISA-Ontology

```
(define-ontology ISA-Ontology
  (creator "IIIA - CSIC")
  (description "Information Search and Aggregation (ISA) Ontology")
  (uses KM-Ontology))

(define-sort Var
  (name any)
  (sort Symbol))

(define-sort FT-Signature-Element
  (name String))

(define-sort FT-Formula
  (name String))

(define-sort (FT-Signature-Element Query-Model)
  (name String "Query-Model")
  (query Query)
  (result  Scored-Item Empty-Set)
  (weight  Number 1)
  (source Source Empty-Set))

(define-sort (FT-Signature-Element Query-Models)
  (name String "Query-Models")
  (q-models Query-Model Empty-Set))

(define-sort (FT-Signature-Element Query)
  (name String "Query")
  (filters Filter Empty-Set))
```

```
(define-sort (Query Domain-Query)
  (name String "Domain-Query")
  (terms Term Empty-Set)
  (category Category))

(define-sort (Query Source-Query)
  (name String "Source-Query")
  (t-filters Filter Empty-Set)
  (source Source))

(define-sort (Source-Query PMID-Query)
  (name String "PMID-Query")
  (pmid String))

(define-sort (Query Source-Queries)
  (name String "Source-Queries")
  (s-queries Source-Query Empty-Set))

(define-sort (FT-Signature-Element Term)
  (name String "Term")
  (term-correlations Term-Correlation Empty-Set)
  (parent Term Empty-Set)
  (children Term Empty-Set))

(define-sort (FT-Signature-Element Category)
  (name String "Category")
  (terms Term-Correlation Empty-Set)
  (filters Filter-Weighting Empty-Set))

(define-sort (FT-Signature-Element Term-Correlation)
  (name String "Term-Correlation")
  (term Term)
  (weight Number 1))

(define-sort (FT-Signature-Element Filter)
  (name String "Filter")
  (attribute String)
  (value String))

(define-sort (FT-Signature-Element Filter-Weighting)
  (name String "Filter-Weighting")
  (filter Filter)
  (weight Number 1))

(define-sort (FT-Signature-Element Item)
  (name String "Item")
  (id String)
  (content any)
  (date number)
  (infoextra string))
```

```
(define-sort (Item Bibliographic-Item)
  (name String "Bibliographic-Item")
  (UID String)
  (Title String)
  (Author String)
  (Publication-date Date)
  (Languages String)
  (Publication-type String))

(define-sort (FT-Signature-Element Scored-Item)
  (name String "Scored-Item")
  (item Item)
  (score Number))

(define-sort (FT-Signature-Element Scored-Items)
  (name String "Scored-Items")
  (s-items Scored-Item Empty-Set))

(define-sort (Sources Source)
  (name String "Source")
  (weight Number 1)
  (search-attributes Attribute-Weighting Empty-Set)
  (basic-attribute Attribute-Weighting)
  (filter-attributes Attribute-Translation Empty-Set)
  (content domain-model))

(define-sort (FT-Signature-Element Sources)
  (name String "Sources")
  (sources Source Empty-Set))

(define-sort (FT-Signature-Element Attribute-Weighting)
  (name String "Attribute-Weighting")
  (attribute String)
  (weight Number))

(define-sort (FT-Signature-Element Attribute-Translation)
  (name String "Attribute-Translation")
  (domain-attribute String)
  (source-attribute String))

(define-sort (FT-Signature-Element Search-Assessment)
  (name String "Search-Assessment")
  (assessment FT-Formula)
  )

(define-sort (FT-Signature-Element Weighted-Pair)
  (name String "Weighted-Pair")
  (weight number)
  (value number))
```

```
(define-sort (FT-Signature-Element Weighted-Pairs)
  (name String "Weighted-Pairs")
  (w-pairs Weighted-Pair Empty-Set))

(define-sort (FT-Signature-Element Item-Info)
  (name String "Item-Info")
  (item Item)
  (pairs Weighted-Pair Empty-Set))

(define-sort (FT-Signature-Element Item-Infos)
  (name String "Item-Infos")
   (item-infos Item-Info Empty-Set))

(define-sort (FT-Signature-Element  Weighting-Function)
  (name "Weighting-Function"))
```

Appendix E

Specification of the ISA-Library

```
(define (Library Wim-Library
  (creator "IIA-CSIC")
  (description "WIM ISA Library with feature terms")
  (uses ORCAS-KM-Ontology ISA-Ontology))


(define (Task :id Information-Search)
  (name "Search")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-CONSULT
      ))))

(define (Task :id PCM-search)
  (name "PCM-Search")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
```

```
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      ASSESS-SEARCH-RESULT
      SATISFY-CONSULT
      ))))

(define (Task :id Modify-search)
  (name "Modify-Search")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-CONSULT
      CHANGE-SCOPE
      )))
  (configuration-options "Configurable On Runtime")
  )


(define (Task :id Critique-search)
  (name "Critique-Search")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (output-roles
   (define (var)
     (name 'assesment)
     (sort Assess-results)))
  (competence
   (define (Competence)
     (postconditions
      ASSESS-SEARCH-RESULT
      )))
  ;(configuration-options "Produces-new-competence")
  )
```

```
(define (Skill :id Search-Assessment)
  (name "Assess-results")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (output-roles
   (define (var)
     (name 'assesment)
     (sort Search-Assessment)))
  (competence
   (define (Competence)
     (postconditions
      ASSESS-SEARCH-RESULT
      )))
  )



(define (Task :id Adapt-query)
  (name "Adapt-query")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 'elab-queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-CONSULT
      CHANGE-SCOPE
      )))
  )

(define (Skill :id Query-generalization)
  (name "Query-generalization")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
```

```
     (name 'elab-queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      GENERALIZE-QUERY
      ELABORATE-CONSULT
      )))
  (knowledge-roles
   Thesaurus)
  )

(define (Skill :id Query-specialization)
  (name "Query-specialization")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 'elab-queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      SPECIALIZE-QUERY
      ELABORATE-CONSULT
      )))
  (knowledge-roles
   Thesaurus)
  )

(define (Task :id Elaborate-query)
  (name "Elaborate-query")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name '?elab-queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-CONSULT
      ))))
```

```
(define (Skill :id Query-expansion-with-thesaurus)
  (name "Query-expansion-with-thesaurus")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 'elab-queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-WITH-THESAURUS
      )))
  (knowledge-roles
   Thesaurus)
  )

(define (Skill :id Exhaustive-query-expansion-with-thesaurus)
  (name "Exhaustive-query-expansion-with-thesaurus")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 'elab-queries)
     (sort Query-models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-WITH-THESAURUS-EXHAUSTIVE
      )))
  (knowledge-roles
   Thesaurus)
  )

(define (Skill :id Query-expansion-with-categories)
  (name "Query-expansion-with-categories")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
```

```
     (name 'elab-queries)
     (sort Query-models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-WITH-CATEGORIES
      )))
  (knowledge-roles
   Categories)
  )


(define (Skill :id Exhaustive-query-expansion-with-categories)
  (name "Exhaustive-query-expansion-with-categories")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 'elab-queries)
     (sort Query-models)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-WITH-CATEGORIES-EXHAUSTIVE
      )))
  (knowledge-roles
   Categories)
  )

(define (Task :id Customise-query)
  (name "Customise-query")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      CUSTOMISE-DOMAIN-QUERY
      )))
  )
```

```
(define (Skill :id Query-customisation)
  (name "Query-customisation")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      NON-EXHAUSTIVE-CUSTOMISATION
      )))
  (knowledge-roles
   Source-Descriptions)
  )


(define (Skill :id Exhaustive-query-customisation)
  (name "Exhaustive-query-customisation")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      EXHAUSTIVE-CUSTOMISATION
      )))
  (knowledge-roles
   Source-Descriptions)
  )

(define (Skill :id Basic-query-customisation)
  (name "Basic-query-customisation")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
```

```
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'queries)
     (sort Query-Models)))
  (competence
   (define (Competence)
     (postconditions
      BASIC-CUSTOMISATION
      )))
  (knowledge-roles
   Source-Descriptions)
  )

(define (Task :id Retrieve)
  (name "Retrieve")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'result)
     (sort Query-Model)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-QUERY
      )))
  )

(define (Skill :id Retrieval)
  (name "Retrieval")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'query)
     (sort Query-Model))
   )
  (output-roles
   (define (var)
     (name 'result)
     (sort Query-Model)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-QUERY
```

```
      )))
  )

(define (Task :id Retrieve-PMID)
  (name "Retrieve-PMID")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'pmid)
     (sort PMID-Query))
   )
  (output-roles
   (define (var)
     (name 'pubmedarticle)
     (sort String)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-PMID
      )))
  )

(define (Skill :id Retrieval-PMID)
  (name "Retrieval-PMID")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'pmid)
     (sort PMID-Query))
   )
  (output-roles
   (define (var)
     (name 'pubmedarticle)
     (sort String)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-PMID
      )))
  )

(define (Task :id Aggregate)
  (name "Aggregate")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-models)
     (sort Query-Models)))
  (output-roles
   (define (var)
```

```
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-ALL
      )))
  )


(define (Task :id Elaborate-items)
  (name "Elaborate-items")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-models)
     (sort Query-Models)))
  (output-roles
   (define (var)
     (name 'item-infos)
     (sort Item-Infos)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-ITEM-INFOS
      )))
  )



(define (Skill :id Items-elaboration)
  (name "Items-elaboration")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'q-model)
     (sort Query-models)))
  (output-roles
   (define (var)
     (name 'item-infos)
     (sort Item-Infos)))
  (competence
   (define (Competence)
     (postconditions
      ELABORATE-ITEM-INFOS
      ))))



(define (Task :id Aggregate-items)
```

```
  (name "Aggregate-items")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'item-inf)
     (sort Item-Info)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-ITEM-INFOS
      )))
  )

(define (Skill :id Arithmetic-mean)
  (name "Arithmetic-mean")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'item-inf)
     (sort Item-Info)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-WITH-ARITHMETIC-MEAN
      )))
  )

(define (Skill :id Weighted-mean)
  (name "Weighted-mean")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'item-inf)
     (sort Item-Info)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-WITH-WEIGHTED-MEAN
```

```
      )))
  )

(define (Skill :id OWA)
  (name "OWA")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'item-inf)
     (sort Item-Info)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-WITH-OWA
      )))
  (knowledge-roles
   Weighting-Function)
  )

(define (Skill :id WOWA)
  (name "WOWA")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'item-inf)
     (sort Item-Info)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Item)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-WITH-WOWA
      )))
  (knowledge-roles
   Weighting-Function))



(define (Task-Decomposer :id Metasearch)
  (name "Metasearch")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
```

```
      (sort Query-Model)))
   (output-roles
    (define (var)
      (name 's-items)
      (sort Scored-Items)))
   (competence
    (define (Competence)
      (postconditions
       SATISFY-CONSULT
       )))
   (subtasks
    Elaborate-query
    Customise-query
    Retrieve
    Aggregate
    ))

(define (Task-Decomposer :id Metasearch-without-elaboration)
   (name "Metasearch-without-elaboration")
   (ontologies ISA-Ontology)
   (input-roles
    (define (var)
      (name 'consult)
      (sort Query-Model)))
   (output-roles
    (define (var)
      (name 's-items)
      (sort Scored-Items)))
   (competence
    (define (Competence)
      (postconditions
       SATISFY-CONSULT
       )))
   (subtasks
    Customise-query
    Retrieve
    Aggregate
    ))


(define (Task-Decomposer :id Aggregation)
   (name "Aggregation")
   (ontologies ISA-Ontology)
   (input-roles
    (define (var)
      (name 'q-models)
      (sort Query-models)))
   (output-roles
    (define (var)
      (name 's-items)
```

```
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      AGGREGATE-ALL
      )))
  (subtasks
   Elaborate-items
   Aggregate-items
   ))

(define (Task-Decomposer :id PCM-metasearch)
  (name "PCM-Metasearch")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      ASSESS-SEARCH-RESULT
      SATISFY-CONSULT
      )))
  (subtasks
   Search
   Critique-search
   Modify-search
   )
  )

(define (Task-Decomposer :id Modify-metasearch)
  (name "Modify-metasearch")
  (ontologies ISA-Ontology)
  (input-roles
   (define (var)
     (name 'consult)
     (sort Query-Model)))
  (output-roles
   (define (var)
     (name 's-items)
     (sort Scored-Items)))
  (competence
   (define (Competence)
     (postconditions
      SATISFY-CONSULT
```

```
    CHANGE-SCOPE
    )))
(subtasks
 Adapt-query
 Customise-query
 Retrieve
 Aggregate
 )
)
```

Appendix F

ORCAS Services

This appendix describes the interaction protocol and the format of the messages to communicate between agents in the ORCAS e-Institution and external agents requesting some ORCAS service.

We distinguish between two types of services:

- information services are used to provide external agents information on the components registered in a library;
- operation services are used by external agents to request institutional agents to perform some action, like configuring a team.

These are the ORCAS operation services:

1. *Brokering*: to obtain a task-configuration satisfying a specification of problem requirements.
2. *Team formation*: to form and instruct a team of agents with the capabilities required by a task-configuration.
3. *Teamwork*: solving a problem by a recently configured team, given a team-identifier.
4. *Cooperative Problem Solving*: this service comprises all the previous services within a single request-inform protocol.

The ORCAS institutional services are accessed through a Personal Assistant (PA) agent, except the informational services, which are provided by the Librarian and can be accessed directly by an external agent. The PA is the mediator between the user and the system, but also between the ORCAS institution and external agents willing to request some of the ORCAS services. The PA agent understands both the ORCAS ontology and the specific application ontology (e.g. the WIM library), freeing the user of knowing them.

In ORCAS the PA role is defined as an external role, since a PA is responsible for interacting with a human user, and needs application specific knowledge in

order to support the user during the problem specification stage. Nonetheless, the ORCAS agent platform provides a generic model of PA that is equipped with the social knowledge required to participate in the ORCAS e-Institution. This PA acts as a broker with respect to other agent willing to use the ORCAS services. In spite of learning the different scenes of the ORCAS institution and communicating other agents directly, an external requester have to communicate only with the PA.

Since the language used by an external agent may differ from the local communication language, there is another kind of agent that mediates between the external agent, and the PA running locally: the FIPA-Mediator. The ORCAS-proxy is responsible for adapting the language used by the external agent to the local language, and viceversa; specifically, it is able to translate messages from the FIPA-ACL to the NOOS ACL, and viceversa. The content is encoded using the FIPA Specification Language (SL) and either XML or RDF to serialize the data.

We focus now on the technical aspects required by external agents to use the ORCAS services through the FIPA-Mediator agent. §F.1 describes the data and protocol for the different services. §F.2 deals with the ontologies and format of the data required by the interaction protocols. §F.4 three contains examples of the FIPA-ACL messages to be interchanged.

F.1 Interaction protocols for the ORCAS services

Figure F.1 shows the interaction protocols for the different ORCAS services using the FIPA style. These diagrams are called Message Sequence Charts (MSC). Each vertical line represents the time running, and horizontal lines represent messages, the vertical rectangles represent agent processing operations, and the rhombuses represent choice points.

Figure F.1.a) shows the MSC for the *Brokering* sevice: The MAS participating in the ORCAS e-institution is configured at the knowledge-level, using the ORCAS-KMF as the Agent Capability Description Language. The external, FIPA-compliant agent, sends a request message with a problem-specification and receives a task-configuration.

Figure F.1.b) shows the MSC for the *Team Formation* service: a team of problem-solving agents is formed and instructed to solve problems according to a task configuration. The external client sends a request message with a task-configuration and receives the identifier of the already formed team.

Figure F.1.c) shows the MSC for the Teamwork service: a problem is solved by a team of problem-solving agents. The team should be previously formed, ensuring that team members have committed to solve the tasks required by the task-configuration in cooperation with the other team-mates. The external FIPA compliant client sends a request message with a problem instance and the identifier of a previously formed team. Data structures for this protocol are

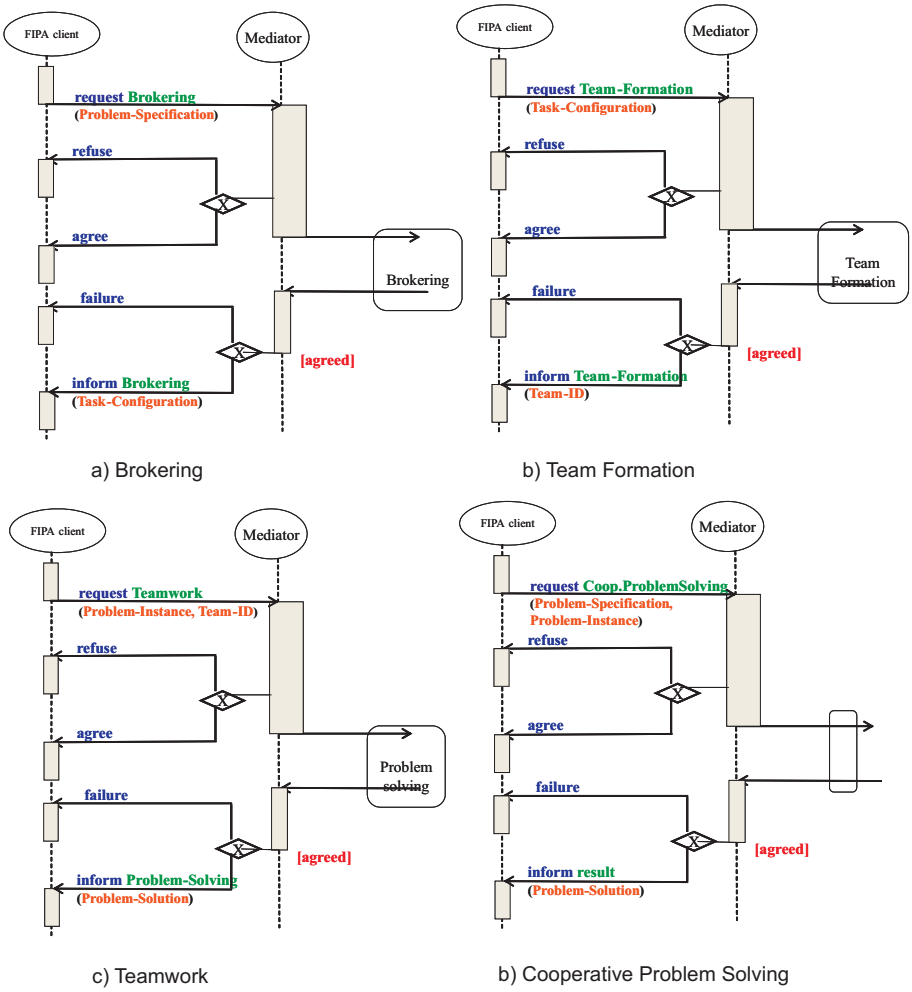


Figure F.1: FIPA Message Sequence Charts for the ORCAS services

described in §F.2.

Figure F.1.d) shows the MSC for the Cooperative Problem-Solving service: The requester sends a request message to the FIPA-Mediator containing a problem specification that is made up of the problem requirements and the problem data. If all goes right with the protocol, at the end the PA sends the result to the FIPA-Mediator, which serializes the data and sends an inform with the result, a set of scored items.

F.2 Data structures and XML format

Problem-Specification			
Parameter	**Description**	**Type**	**Cardinality**
task-name	Name of the task to be configured	String	Single
preconditions	Constraints over the inputs of the task	Signature-element	Multiple
postconditions	Constraints over the output of the task, relations between ouput and input	Signature-element	Multiple
input-roles	Inputs of task	Formula	Multiple
knowledge-roles	Domain models of the knowledge to be used	Signature-element	Multiple

Table F.1: Problem specification

Task-Configuration			
Parameter	**Description**	**Type**	**Cardinality**
task-name	Name of the task being configured	Symbol	Single
input-roles	Input roles of the task	Signature-element	Multiple
capability-configuration	A configuration for a capability bound to the task.	Capability-configuration	Single

Table F.2: Task configuration

This is XML grammar for the content language to be used when accessing ORCAS services.

```
Problem-Specification:=
   <problem-specification>
```

TD-Configuration (is a Capability-Configuration)			
Parameter	**Description**	**Type**	**Cardinality**
capability-name	Name of the capability being configured	Symbol	Multiple
input-roles	Input roles of the Task Decomposer	Signature-Element	Multiple
subtasks-configuration	Configuration for the subtasks	Task-configuration	Multiple
operational-description	Intermediate roles and program description	Operational-Description	Single

Table F.3: Task-Decomposer configuration

Skill-Configuration (is a Capability-Configuration)			
Parameter	**Description**	**Type**	**Cardinality**
capability-name	Name of the capability being configured	Symbol	Multiple
input-roles	Input roles of the Skill	Signature-Element	Multiple
domain-models	Names of the domain-models to be used	Symbol	Multiple
knowledge-roles	Domain models to be used by the capability	Signature-element	Multiple

Table F.4: Skill configuration

```
        <task-name> Symbol</task-name>
        <preconditions>Formula*</preconditions>
        <postconditions>Formula*</postconditions>
        <inputs>Signature-Element*</preconditions>
        <knowledge-roles>Signature-Element*</ knowledge-roles >
    </problem-specification>

Task-Configuration :=
    <task-configuration>
        <task-name> Symbol </task-name>
        <capability-configuration>Capability-Configuration</capability-configuration>
    </task-configuration>

Capability-Configuration:= Task-Decomposer-Configuration |
Skill-Configuration

Task-Decomposer-Configuration :=
    <task-decomposer-configuration>
        <capability-name> Symbol </capability-name>
        <subtasks-configuration>Task-Configuration</subtasks-configuration>
    </Task-Decomposer-configuration>

Skill-Configuration :=
    <skill-configuration>
        <capability-name> Symbol </task-id>
        <domain-models>Symbol*</domain-models>
    </skill-configuration>

Formula:= <formula>String</formula>

Signature-Element:= <signature-element>String</signature-element>

Symbol:= String
```

F.3 ORCAS services in the WIM application

This section describes the aspects of the WIM ontology to be used by external clients requesting for some of the ORCAS services in the WIM application. Only those concepts used to communicate with the FIPA-Mediator are considered, which are basically keyword-based queries and scored-items, while other concepts concepts not required by an external agent to communicate with WIM are omitted..

```
User-Consult:=
        <user-consult>
            <query>Domain-Query </query>
            <sources>Source*</source>
        </user-consult>

Domain-Query :=
        <query>
            <keywords> Keyword* </keywords>
            <filters>Filter*</filters>
            <category>Category</category>
        </query>

Filter:=
        <filter>
            <attribute>Attribute</attribute>
            <value>String</value>
        </filter >
```

```
Source:=    <source>Symbol</source>

Keyword:=   <keyword>Symbol</keyword>

Scored-Item :=
        <scored-item>
            <item>Item</item>
            <score>Number</score>
        </scored-item>

Item :=
        <item>
            <id>String </id>
            <title>String</title>
            <author>String</author>
        </item>

Source := Pubmed | Medline-IGM | Healthstar-IGM | ISOCO

Category:=  Good-Evidence | Medium-Evidence |
            Poor-Evidence | Evidence

Attribute:= Author Name | Begin Year | End Year |
            Publication Type | Language | Journal
```

F.4 FIPA examples

F.4.1 Brokering

```
(request
    :sender     (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
    :receiver   (set (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS))
    :reply-with     configuration-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (action
        (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
        (Brokering
        :problem-specification
        (Problem-Specification
        :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
        <problem-specification>
            <task-name> Search </task-name>
            <postconditions>
                <formula>Satisfy-Consult</formula>
                <formula>Non-Exhaustive-Customization</formula>
                <formula>Aggregate-With-Arithmetic-mean</formula>
            </postconditions>
        <input-roles>
            <signature-element>Query-Model</signature-element>
        </input-roles>
        <knowledge-roles>
             <signature-element>Source-Descriptions</signature-element>
        </knowledge-roles>
        </problem-specification>))))

(agree ...)

(inform
    :sender     (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS)
    :receiver   (set (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE))
```

```
    :reply-with     configuration-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (result
          (action )
          (Task-configuration
        :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?>
        :value  <task-configuration></task-configuration>))))
```

F.4.2 Team formation

```
 (request
    :sender     (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
    :receiver   (set (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS))
    :reply-with     team-request 18237
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
    (action
        (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS)
        (Team-Formation
        :task-configuration
        (Task-Configuration
        :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
        <task-configuration>...</task-configuration>))))))

(agree ...)

(inform
    :sender     (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS)
    :receiver   (set (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE))
    :reply-with  team-request 18237
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (result
          (action ... )
            (Team-ID
            :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
            <team-id> Team-18237 </team-id>))))
```

F.4.3 Problem-Solving

```
(request
    :sender     (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
    :receiver   (set (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS))
    :reply-with     problem-solving-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (action
        (agent-identifier :name \wim\-Proxy@wim.iiia.csic.es:7778/NOOS)
```

```
        (Problem-Solving
           :problem-instance
           (Problem-Instance
           :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
           <user-consult>
               <query>
                   <keywords>
                       <keyword>Ofloxacin</keyword>
                       <keyword>Ofloxacin</keyword>
                       <keyword>Guidelines</keyword>
                   </keywords>
               </query>
               <sources>
                   <source>Pubmed</source>
                   <source>ISOCO<source>
               </sources>
            </user-consul
          :team-ID
          (Team-ID:
             :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
             <team-id> Team-18237 </team-id>))))

(agree ...)

(inform
    :sender     (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS)
    :receiver   (set (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE))
    :reply-with     problem-solving-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (result
        (action ...)
        (Problem-Solution
        :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?>
        :value
        <scored-items>...</scored-items>))))
```

F.4.4 Cooperative Problem-Solving

```
(request
    :sender     (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
    :receiver   (set (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS))
    :reply-with     configuration-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (action
        (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE)
        (FullProblemSolving
        :problem-specification
           (ProblemSpecification
           :encoding  <xml? version="1.0" encoding ="ISO-8859-1"> :value
           <problem-specification>
               <task-name> Search </task-name>
               <postconditions>
                   <formula>Satisfy-Consult</formula>
                   <formula>Non-Exhaustive-Customization</formula>
                   <formula>Aggregate-With-Arithmetic-mean</formula>
               </postconditions>
           <input-roles>
```

```
            <signature-element>Query-Model</signature-element>
          </input-roles>
          <knowledge-roles>
               <signature-element>Source-Descriptions</signature-element>
          </knowledge-roles>
          </problem-specification>)
        :problem-instance
           (Problem-Instance
          :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
          <user-consult>
              <query>
                  <keywords>
                      <keyword>Ofloxacin</keyword>
                      <keyword>Ofloxacin</keyword>
                      <keyword>Guidelines</keyword>
                  </keywords>
              </query>
              <sources>
                  <source>Pubmed</source>
                  <source>ISOCO<source>
              </sources>
          </user-consul
         :team-ID
         (Team-ID:
            :encoding  <xml? version='1.0' encoding ='ISO-8859-1'?> :value
            <team-id> Team-18237 </team-id>))))

(agree ...)


(inform
    :sender     (agent-identifier :name WIM-Proxy@wim.iiia.csic.es:7778/NOOS)
    :receiver   (set (agent-identifier :name uva-agent@a1136.fmg-uva-nl:1099/JADE))
    :reply-with     problem-solving-request18236
    :encoding   String
    :language   FIPA-SL0
    :ontology   WIM-Ontology
    :protocol   FIPA-request
    :conversation-id    configuration18236
    :content
       (result
         (action ... )
         (Problem Solution: :encoding  <xml? version="1.0"
            encoding ="ISO-8859-1"?> :value
            <scored-items>
                <scored-item>
                    <item>
                       <Identifier>PMID1756688</Identifier>
                       <Title>Treatment of lower respiratory infections in outpatients
                       with ofloxacin compared with erythromycin.</title>
                       <Author>Peugeot RL, Lipsky BA, Hooton TM, Pecoraro RE.</Author>
                    </item>
                    <score>0.02</score>
                </scored-item>
                <scored-item>
                    <item>
                       <Identifier>PMID1864291</Identifier>
                       <Title>Role of quinolones in the treatment of bronchopulmonary infections,
                        particularly pneumococcal and community-acquired pneumonia..</Title>
                       <Author>Thys JP, Jacobs F, Byl B.</Author>
                    </item>
                    <score>0.02</score>
                </scored-item>
            </scored-items>))))
```

F.5 The Personal Assistant

The Personal Assistant (PA) mediates between the human user or an external agent and the other agents (institutional and PSAs). The PA helps the user specifying a problem by using the user's domain ontology, and avoids him knowing technical details like the agent communication language and the interaction protocols (scenes in the ORCAS e-Institution underlying the WIM application). Specifically, the PA is able to transform the user specification of the problem into a problem specification using the ORCAS Agent Capability Description Language (ACDL). This problem specification contains the problem requirements to be used by the institutional agents to form a new team of agents that is able to solve the problem at hand according to the requirements specified by the user.

The PA brings an added value to the WIM services, for it is able to organize the user tasks as a collection of interests and goals and schedule them to update the results periodically. An interest refers to a topic or a subject the user is interested in while goals are specific issues the user wants to search information on and are represented as specific queries to look up on bibliographic databases. An interest is specified as a collection of goals together with a set of preferences (problem requirements, configuration strategy, scheduling options, etc.), and each goal is specified as a consultation (keywords, filters, category, information sources, etc.), plus a set of preferences. The preferences of a goal are inherited from the interest, but they can be refined for any particular goal. We will show some examples of the functionality offered by the PA to human users through a Web interfaced defined for the WIM application. In order to provide a Web interface to the application, we have connected the ORCAS e-Institution to an *http server* through a pseudo agent called the *www-mediator*, as showed in Figure F.2.

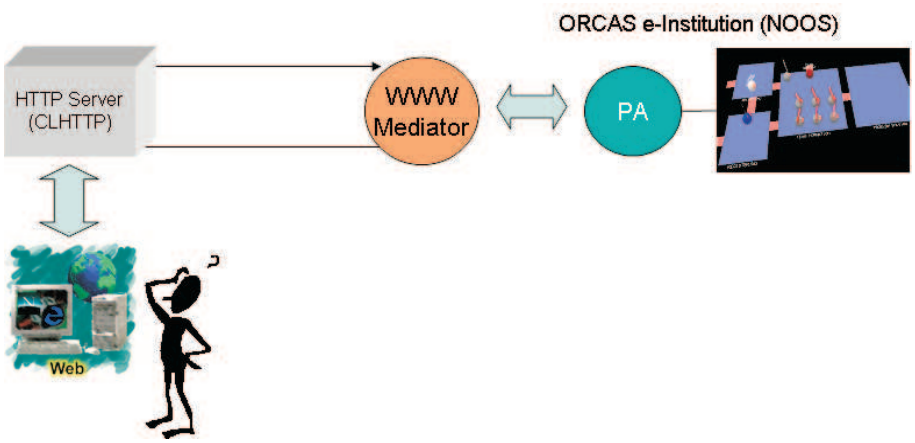


Figure F.2: Web interface to WIM

The http-server calls a function of the www-mediator using data obtained

from a Web form, and builds a results http page using the data returned by the www-mediator after communicating with the PA. Therefore, the www-mediator can be seen as an adaptor or wrapper that agentifies the http-server so as to allow the PA to interact with it. The www-mediator operates by accepting function calls from the http-server, translating the Web data format to the data format used in the NOOS agent platform, and communicating with the PA using the agent communication language.

Figure F.3 shows an example of a user interested in three topics: *Cannabis*, *AIDS*, and *Pneumonia prognosis and therapy*. On the right side, the figure shows the specification of the interest called *AIDS*, which has two goals: *AIDS classification* and *AIDS therapy*.

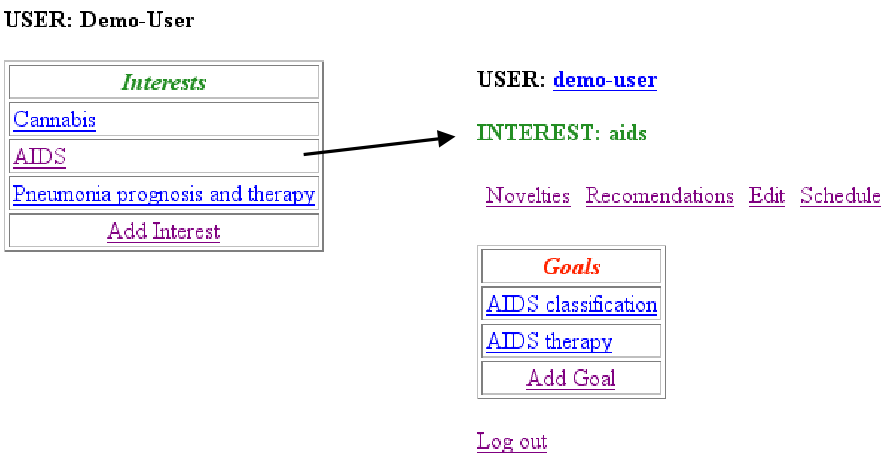


Figure F.3: Managing Interests and goals

Figure F.4 shows the interface used to edit one goal. The user can specify up to three keywords to characterize the subject of the information search, together with a category from the knowledge base on Evidence Based Medicine (EBM). Categories can be used to enrich the user's query and rank the information retrieved in form of scored items according to some of the subjects typically required by EBM practitioners (e.g. references on diagnosis and therapy, or clinical protocols), or by a desired degree of evidence for the references to be retrieved (this is defined by a three-levels ordinal scale: Good, Medium, and Poor Evidence Quality). The user may also specify requirements, such as the type of query elaboration desired and the aggregation operator preferred. Search filters like publication date periods (Begin-Year and End-Year), author name, publication type, language, and journal.

In addition, goals can be scheduled by the PA to repeat the consultation periodically, allowing the user to automatically update the results for each goal. Figure F.5 shows the interface used to schedule the execution of one goal. A goal

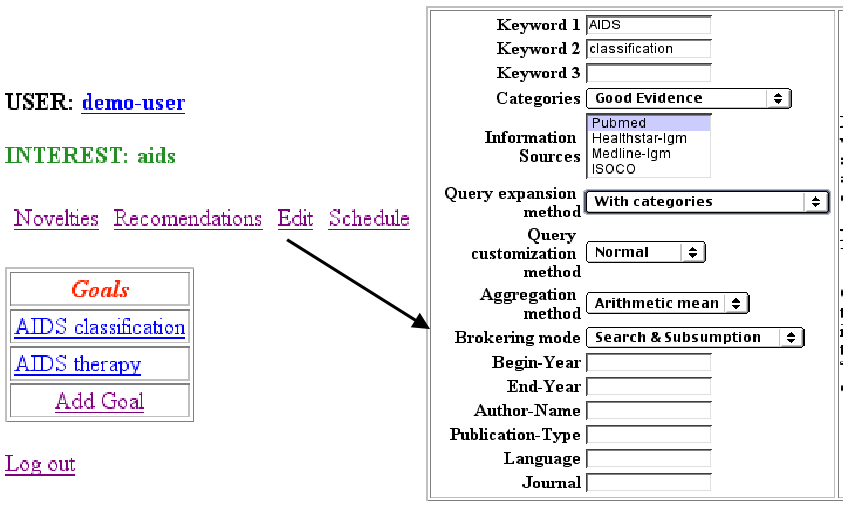


Figure F.4: Goal editing

can be defined as permanent or volatile (executed only once). When it is stated as permanent, the user can specify the typical schedulling options for periodic execution of tasks, such as time, week-day and month-day.

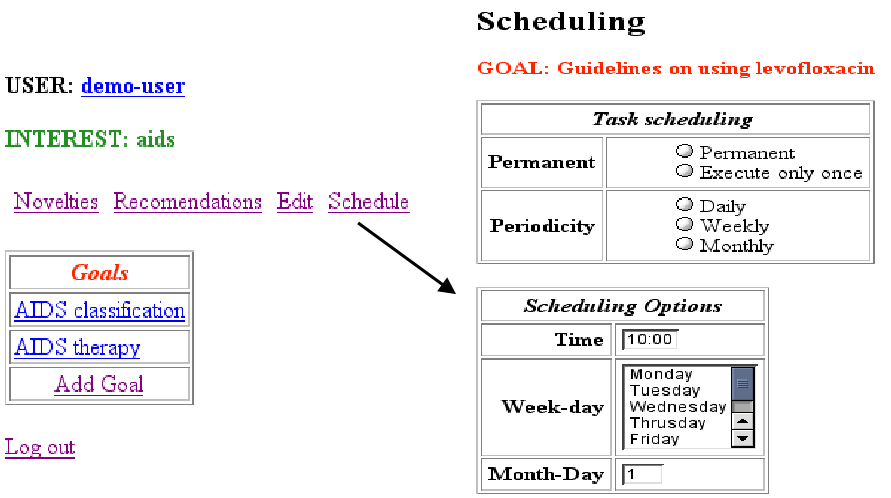


Figure F.5: Scheduling

Appendix G

Glossary of abbreviations

ACDL Agent Capability Description Language

CPS Cooperative Problem Solving (a process involving agents)

DAML DARPA Agent Markup Language

EBM Evidence-Based Medicine

HTN Hierarchical Task Network (a type of planning)

IGM Internet Grateful Med (a Web-based search-engine)

ISA Information Search Library

KB Knowledge-Broker (an ORCAS agent role)

KMF Knowledge Modelling Framework

KMO Knowledge Modelling Ontology

MAS Multi Agent Systems

MSC Message Chart Diagram

MeSH Medical Subject Headings

ORCAS Open, Reusable and Configurable multi-Agent Systems

OWA Ordered Weighted Average

PA Personal Assistant (an ORCAS agent role)

PSA Problem-Solving Agent (an ORCAS agent role)

PSM Problem Solving Method

Pubmed Public Medline (a Web-based search-engine)

SWS Semantic Web Services

TB Team-Broker (an ORCAS agent role)

TMD Task-Method-Domain (a model used in Knowledge Modelling)

WIM Web Information Mediator (an ORCAS based application to search information in the Internet)

WOWA Weighted OWA

Bibliography

[Abasolo and Gómez, 2000] Abasolo, C. and Gómez, M. (2000). Melisa: An ontology-based agent for information retrieval in medicine. In *Proceedings of the First International Workshop on the Semantic Web (SemWeb2000)*, pages 73–82.

[Aitken et al., 1998] Aitken, S., Filby, I., Kingston, J., and Tate, A. (1998). Capability descriptions for problem-solving methods. Submitted to the European Conference on AI.

[Ankolekar et al., 2002] Ankolekar, A., Huch, F., and Sycara, K. P. (2002). Concurrent semantics for the web services specification language DAML-s. In *Coordination Models and Languages*, pages 14–21.

[Arcos, 1997] Arcos, J. L. (1997). *The Noos representation language*. Monografies del iiia, Universitat Politècnica de Catalunya.

[Arcos, 2001] Arcos, J. L. (2001). T-air: A case-based reasoning system for designing chemical absorption plants. In Aha, D. W. and Watson, I., editors, *Case-Based Reasoning Research and Development*, number 2080 in Lecture Notes in Artificial Intelligence, pages 576–588. Springer-Verlag.

[Arcos and López de Mántaras, 1997] Arcos, J. L. and López de Mántaras, R. (1997). Perspectives: a declarative bias mechanism for case retrieval. In Leake, D. and Plaza, E., editors, *Case-Based Reasoning. Research and Development*, number 1266 in Lecture Notes in Artificial Intelligence, pages 279–290. Springer-Verlag.

[Arcos et al., 1998] Arcos, J. L., López de Mántaras, R., and Serra, X. (1998). Saxex : a case-based reasoning system for generating expressive musical performances. *Journal of New Music Research*, 27 (3):194–210.

[Arens et al., 1993] Arens, Y., Chee, C. Y., Hsu, C.-N., and Knoblock, C. A. (1993). Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158.

[Armengol and Plaza, 1997] Armengol, E. and Plaza, E. (1997). Induction of feature terms with INDIE. In van Someren, M. and Widmer, G., editors, *Euro-*

pean Conference on Machine Learning, Lecture Notes in Artificial Intelligence. Springer-Verlag.

[Armengol and Plaza, 2001] Armengol, E. and Plaza, E. (2001). Similarity assessment for relational CBR. In *Case-Based Reasoning Research and Development*, volume 2080 of *Lecture Notes in Computer Science*, pages 44–58.

[Atkinson, 1997] Atkinson, S. (1997). *Engineering Software Library Systems*. PhD thesis, School of Information Technology, University of Queensland, 1997. To appear.

[Bansal and Vidal, 2003] Bansal, S. and Vidal, J. M. (2003). Matchmaking of web services based on the DAML-S service model. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*.

[Barbuceanu and Fox, 1995] Barbuceanu, M. and Fox, M. (1995). COOL: a language for describing coordination in multi-agent systems. In *Proceedings of the First International Conference in Multi-Agent Systems ICMAS'95*, pages 17–24. AAAI Press.

[Bastide et al., 1999] Bastide, R., Sy, O., and Palanque, P. (1999). Formal specification and prototyping of corba systems. In *Proceedings of 13th European Conference on Object-Oriented Programming*, volume 1628 of *Lecture Notes in Computer Science*, pages 474–494. Springer-Verlag.

[Beckers et al., 1994] Beckers, R., Holland, O., and Deneubourg, J. (1994). From local actions to global tasks: Stigmergy and collective robotics. In *Artificial Life IV. 4th International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press.

[Benjamins, 1993] Benjamins, R. (1993). *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam.

[Benjamins, 1997] Benjamins, R. (1997). Problem-solving methods in cyberspace. In *Proceedingsof the Workshop on Problem-Solving Methods for Knowledge-based Systems of IJCAI*.

[Benjamins et al., 1996a] Benjamins, R., Fensel, D., and Chandrasekaran, B. (1996a). PSMs do IT. Summary of track on Sharable and Reusable Problem-Solving Methods. KAW'96.

[Benjamins et al., 1999] Benjamins, R., Wielinga, B., Wielemaker, J., and Fensel, D. (1999). Towards brokering problem-solving knowledge on the internet. In *Knowledge Acquisition, Modeling and Management*, pages 33–48.

[Benjamins et al., 1996b] Benjamins, V., de Barros, L., and Andre, V. (1996b). Constructing planners through problem-solving methods. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*.

[Benjamins et al., 1998] Benjamins, V., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, B., Schreiber, G., Zdrahal, Z., and Decker, S. (1998). Ibrow3: An intelligent brokering service for knowledge-component reuse on the world-wide web.

[Benjamins et al., 1996c] Benjamins, V. R., Fensel, D., and Straatman, R. (1996c). Assumptions of problem-solving methods and their role in knowledge engineering. In *European Conference on Artificial Intelligence*, pages 408–412.

[Berners-Lee et al., 1999] Berners-Lee, T., Fischetti, M., and Dertouzos, M. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, San Francisco.

[Biggerstaff and Perlis, 1989] Biggerstaff, T. J. and Perlis, A. J., editors (1989). *Software Reusability*. ACM press.

[Birmingham et al., 1995] Birmingham, W. P., Durfee, E. H., Mullen, T., and Wellman, M. P. (1995). The distributed agent architecture of the university of michigan digital library (extended abstract). In *AAAI Spring Symposium on Information Gathering*.

[Bond and Gasser, 1988a] Bond, A. and Gasser, L., editors (1988a). *Readings in distributed Artificial Intelligence*. Morgan Kaufmann Publishers.

[Bond and Gasser, 1988b] Bond, A. H. and Gasser, L. (1988b). An analysis of problems and research in dai. In Bond, A. H. and Gasser, L., editors, *Readings in Distributed Artificial Intelligence*, pages 61–70. Kaufmann, San Mateo, CA.

[Börstler, 1995] Börstler, J. (1995). Feature-oriented classification for software reuse. In *Proceedings Seventh International Conference of Software Engineering and Knowledge Engineering*.

[Bratman, 1988] Bratman, M. E. (1988). Plans and resource bounded practical reasoning. *Computational Intelligence*, 4:249–355.

[Bratman, 1990] Bratman, M. E. (1990). What is intention? In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 15–31. MIT Press, Cambridge, MA.

[Bratman, 1992] Bratman, M. E. (1992). Shared cooperative activity. *Philosophical Review*, 101:327–341.

[Bratman et al., 1991] Bratman, M. E., Israel, D., and Pollack, M. (1991). Plans and resource-bounded practical reasoning. In Cummins, R. and Pollock, J. L., editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts.

[Brazier et al., 1997] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J. (1997). DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94.

[Brazier et al., 2002] Brazier, F. M. T., Jonker, C. M., and Treur, J. (2002). Principles of component-based design of intelligent agents. *Data Knowledge Engineering*, 41(1):1–27.

[Breuker, 1994] Breuker, J. (1994). A suite of problem types. In Breuker, J. and Van de Velde, W. W., editors, *CommonKADS Library for Expertise Modeling*, volume 21 of *Frontiers in Artificial Intelligence and Applications*, pages 57–88. IOS-Press.

[Breuker and Van de Velde, 1994] Breuker, J. and Van de Velde, W., editors (1994). *COMMONKADS Library for Expertise Modelling*. IOS Press.

[Brown and Wallnau, 1996] Brown, A. W. and Wallnau, K. C. (1996). Engineering of component-based systems. In *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*, pages 7–15. IEEE Computer Society Press.

[Bryson et al., 2002] Bryson, J. J., Martin, D., McIlraith, S., and Stein, L. A. (2002). Agent-based composite services in daml-s: The behavior-oriented design of an intelligent semantic web. In Zhong, N., Liu, J., and Yao, Y., editors, *Web Intelligence*. Springer-Verlag.

[Buchanan et al., 1983] Buchanan, B. et al. (1983). Constructing an expert system. In Hayes-Roth, F., Waterman, D., and D.Lenat, editors, *Building Expert Systems*. Addison-Wesley.

[Burmeister, 1996] Burmeister, B. (1996). Models and methodology for agent-oriented analysis and design. In Fischer, K., editor, *Proceedings of the Workshop on Agent-Oriented Programming and Distributed Systems*. DFKI Document D-96-06.

[Bussler et al., 2002] Bussler, C., Maedche, A., and Fensel, D. (2002). A conceptual architecture for semantic web enabled web services. *SIGMOD Record*, 31(4):24–29.

[Butler and Duke, 1998] Butler, S. and Duke, R. (1998). Defining composition operators for object interaction. *Object Oriented Systems*, 5(1):1–16.

[Bylander and Chandrasekaran, 1988] Bylander, T. and Chandrasekaran, B. (1988). Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In Gaines, B. and Boose, J., editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press.

[Cammarata et al., 1983] Cammarata, S., MacArthur, D., and Steeb, R. (1983). Strategies of cooperation in distributed problem solving. In *Proceedins of the Eighth International Joint Conference on Artificial Intelligence*.

[Canal et al., 2001] Canal, C., Fuentes, L., Pimentel, E., Troya, J. M., and Vallecillo, A. (2001). Extending CORBA interfaces with protocols. *The Computer Journal*, 44(5):448–462.

[Carbonell, 2000] Carbonell, J. (2000). ISMIS invited talk.

[Cardoso and Sheth, 2002] Cardoso, J. and Sheth, A. P. (2002). Semantic e-workflow composition. Technical report, LSDIS Lab, Department of Computer Science, University of Georgia.

[Chandrasekaran, 1986] Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1:23–30.

[Chandrasekaran, 1987] Chandrasekaran, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. In *International Joint Conference on Artificial Intelligence*, pages 1183–1192.

[Chandrasekaran, 1990] Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI Magazine*, 11(4):59–71.

[Chandrasekaran and Johnson, 1993] Chandrasekaran, B. and Johnson, T. (1993). Generic tasks and task structures: History, critique and new directions. In David, J., Krivine, J., and Simmons, R., editors, *Second Generation Expert Systems*, pages 239–280. Springer-Verlag.

[Chandrasekaran et al., 1992] Chandrasekaran, B., Johnson, T., and Smith, J. (1992). Task structure analysis for knowledge modeling. *Communications of the ACM*, 33(9):124–136.

[Chandrasekaran et al., 1998] Chandrasekaran, B., Josephson, J., and Benjamins, R. (1998). The ontology of tasks and methods. In *In Proceedings of the 11th Knowledge Acquisition Modeling and Management Workshop, KAW'98, Banff, Canada, April 1998.*

[Chawathe et al., 1994] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. (1994). The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18.

[Cheyer and Martin, 2001] Cheyer, A. and Martin, D. (2001). The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148. OAA.

[Clancey, 1989] Clancey, W. (1989). The knowlede level reinterpreted. *Machine Learning*, 4:285–291.

[Clement and Durfee, 1999] Clement, B. J. and Durfee, E. H. (1999). Top-down search for coordinating the hierarchical plans or multiple agents. In Etzioni, O., Muller, J. P., and Bradshaw, J. M., editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 252–259, Seattle, WA, USA. ACM Press.

[Clements, 1996] Clements, P. C. (1996). From subroutines to subsystems: Component-based software development. In *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*, pages 3–6. IEEE Computer Society Press.

[Cohen and Levesque, 1990] Cohen, P. R. and Levesque, H. J. (1990). Persistence, intention, and commitment. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 33–69. MIT Press, Cambridge, MA.

[Cohen and Levesque, 1991] Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Nous*, 25(4):487–512.

[Decker, 1996] Decker, K. (1996). TAEMS: A Framework for Environment Centered Analysis and Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science.

[Decker and Lesser, 1995] Decker, K. and Lesser, V. R. (1995). Designing a family of coordination algorithms. In Lesser, V., editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, San Francisco, CA, USA. The MIT Press: Cambridge, MA, USA.

[Decker et al., 1997a] Decker, K., Pannu, A., Sycara, K., and Williamson, M. (1997a). Designing behaviors for information agents. In *Proceedings of the 1st International Conference on Autonomous Agents*, pages 404–412. ACM Press.

[Decker et al., 1997b] Decker, K., Sycara, K., and Williamson, M. (1997b). Middle-agents for the internet. In *Proceedings the 15th International Joint Conference on Artificial Intelligence*, pages 578–583.

[Decker et al., 1996] Decker, K., Williamson, M., and Sycara, K. (1996). Matchmaking and brokering. In *Proceedings of the 2nd International Conference in Multi-Agent Systems*.

[Decker and Lesser, 1992] Decker, K. S. and Lesser, V. R. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346.

[Dellarocas, 2000] Dellarocas, C. (2000). Contractual agent societies negotiated shared connote and social control in open multi-agent systems. In *Proceedings of the Workshop on Norms and Institutions in Multi-Agent Systems, ICMAS'02*.

[Dellarocas and Klein, 1999] Dellarocas, C. and Klein, M. (1999). Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In *Proceedings ACM Conference on Electronic Commerce (EC-99)*.

[Dignum et al., 2001] Dignum, F., Dunin-Keplicz, B., and Verbrugge, R. (2001). Agent theory for team formation by dialogue. In *Intelligent Agents VII, Agent Theories Architectures and Languages*, volume 1986 of *Lecture Notes in Artificial Intelligence*, pages 150–166. Springer-Verlag.

[Dignum et al., 2002] Dignum, V., Meyer, J.-J., Weigand, H., and Dignum, F. (2002). An organization-oriented model for agent societies. In *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications*.

[d'Inverno et al., 1997] d'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., de Rijke, M., Ryan, M., and Wooldridge, M. (1997). Formalisms for multi-agent systems. *Knowledge Engineering Review*, 12(3).

[d'Inverno et al., 1998] d'Inverno, M., Kinny, D., and M.Luck (1998). Interaction protocols in agentis. In *Proceedings of the Third International Conference on Multi-Agent Systems ICMAS'98*, pages 112–119.

[Doran and Palmer, 1995] Doran, J. and Palmer, M. (1995). The eos project: modelling prehistoric sociocultural trajectories. In *Aplicaciones informaticas en Arqueologia: Teoria y Sistemas. Proceedings of First International Symposium on Computing and Archaeology (1991)*, volume 1.

[Doran et al., 1997] Doran, J., S.Franklin, Jennings, N., and T.J.Norman (1997). On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):1–6. Panel Discussion at the First UK Workshop on Foundations of Multi-Agent Systems.

[Duke et al., 1991] Duke, R., King, P., Rose, G., and Smith, G. (1991). The object-z specification language. Technical report, Department of Computer Science, University of Queensland.

[Durfee, 1988] Durfee, E. H. (1988). *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers.

[Durfee and Lesser, 1989] Durfee, E. H. and Lesser, V. (1989). Negotiating task decomposition and allocation using partial global planning. In Gasser, L. and Huhn, M., editors, *Distributed Artificial Intelligence Volume II*, pages 229–244. Pitman Publishing.

[Durfee et al., 1998] Durfee, E. H., Mullen, T., Park, S., Vidal, J. M., and Weinstein, P. (1998). The dynamics of the UMDL service market society. In Klusch, M. and Weiss, G., editors, *Cooperative Information Agents II*, Lecture Notes in Artificial Intelligence, pages 55–78. Springer.

[Ephrati and Rosenschein, 1996] Ephrati, E. and Rosenschein, J. S. (1996). Deriving consensus in multiagent systems. *Artificial Intelligence*, 87(1-2):21–74.

[Erickson, 1996a] Erickson, T. (1996a). An agent-based framework for interoperability. In Bradshaw, J. M., editor, *Software Agents*. AAAI Press.

[Erickson, 1996b] Erickson, T. (1996b). Designing agents as if people mattered. In Bradshaw, J. M., editor, *Software Agents*. AAAI Press.

[Eriksson et al., 1995] Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., and Musen, M. A. (1995). Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326.

[Erol, 1995] Erol, K. (1995). *Hierarchical Task Network Planning: Formalization, Analysis and Implementation*. PhD thesis, University of Maryland.

[Erol et al., 1994] Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA. AAAI Press/MIT Press.

[Esteva, 1997] Esteva, M. (1997). *Electronic Institutions: From Specification to Development*. PhD thesis, Universitat Autnoma de Barcelona.

[Esteva et al., 2002a] Esteva, M., de la Cruz, D., and Sierra, C. (2002a). Islander: an electronic institutions editor. In *Proceedings 1th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1045–1052.

[Esteva et al., 2002b] Esteva, M., Padget, J., and Sierra, C. (2002b). Formalizing a language for institutions and norms. In *Intelligent Agents VIII: Lecture Notes in Artificial Intelligence*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 348–366. Springer-Verlag.

[Esteva et al., 2001] Esteva, M., Rodriguez, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). On the formal specifications of electronic institutions. In *Agent-mediated Electronic commerce. The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Artificial Intelligence*, pages 126–147.

[Euzenat, 2001] Euzenat, J. (2001). An infrastructure for formally ensuring interoperability in a heterogeneous semantic web. In *Proc. 1st internationnal on semantic web working symposium (SWWS), Stanford (CA US)*, pages 345–360.

[Feinstein and Horwitz, 1997] Feinstein, A. and Horwitz, R. (1997). Problems in the evidence of evidence-based medicine. *American Journal of Medicine*, 103:529–535.

[Fensel, 1997a] Fensel, D. (1997a). An ontology-based broker: Making problem-solving method reuse work. In *Proceedings Workshop on Problem-solving Methods for Knowledge-based Systems at IJCAI'97*.

[Fensel, 1997b] Fensel, D. (1997b). The tower-of-adapter method for developing and reusing problem-solving methods. In *Knowledge Acquisition, Modeling and Management*, pages 97–112.

[Fensel et al., 1998a] Fensel, D., Angele, J., and Studer, R. (1998a). The knowledge acquisition and representation language karl. *IEEE Transcactions on Knowledge and Data Engineering*, 10(4):527–550.

[Fensel and Benjamins, 1998a] Fensel, D. and Benjamins, R. (1998a). The role of assumptions in knowledge engineering. *International Journal of Intelligent Systems*.

[Fensel and Benjamins, 1998b] Fensel, D. and Benjamins, V. (1998b). Key issues for automated problem-solving methods reuse. In *Proceedings 13th European Conference on Artificial Intelligence*.

[Fensel et al., 1999] Fensel, D., Benjamins, V., Motta, E., and Wielinga, B. (1999). UPML: A framework for knowledge system reuse. In *International Joint Conference on AI*, pages 16–23.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The web service modelling framework. Technical report, Vrijer Universiteit Amsterdam.

[Fensel et al., 1997] Fensel, D., Decker, S., Motta, E., and Zdrahal, Z. (1997). Using ontologies for defining task, problem-solving methods and their mappings. In *Proceedings European Knowledge Acquisition Workshop*, Lecture Notes in Artificial Intelligence.

[Fensel et al., 1998b] Fensel, D., Groenboom, R., and de Lavalette, G. (1998b). Modal change logic (mcl): Specifying the reasoning of knowledge-based systems. *Data and Knowledge Engineering*, 26(3):243–269.

[Fensel et al., 2000] Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. C. A. (2000). OIL in a nutshell. In *Proceedings of the European Knowledge Acquisition, Modeling and Management Conference (EKAW)*, Lecture Notes in Artificial Intelligence, pages 1–16. Springer-Verlag.

[Fensel and Motta, 2001] Fensel, D. and Motta, E. (2001). Structured development of problem solving methods. *Knowledge and Data Engineering*, 13(6):913–932.

[Fensel and Straatman, 1996] Fensel, D. and Straatman, R. (1996). Problem-solving methods: Making assumptions for efficiency reasons. *Lecture Notes in Computer Science*, 1076:17–??

[Finin et al., 1994] Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an Agent Communication Language. In Adam, N., Bhargava, B., and Yesha, Y., editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA. ACM Press.

[Fink, 1998] Fink, E. (1998). How to solve it automatically: Selection among problem-solving methods. In *Proceedings of AIPS*, pages 128–136.

[FIPA, 2002] FIPA (2002). FIPA contract net interaction protocol specification.

[FIPA, 2003] FIPA (2003). Agent Communication Language Specifications. http://www.fipa.org/repository/aclspecs.html.

[Fischer et al., 1995] Fischer, B., Kievernagel, M., and Struckmann, W. (1995). VCR: A VDM-based software component retrieval tool. In *Proc. ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice.*

[Fisher et al., 1997] Fisher, M., Muller, J., Schroeder, M., Staniford, G., and Wagner, G. (1997). Methodological foundations for agentbased systems. *Knowledge Engineering Review*, 12(3):323–329.

[Franklin,] Franklin, S. Coordination without communication. http://www.msci.memphis.edu/ franklin/coord.html.

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A. (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany. Springer-Verlag.

[Garland and Perry, 1995] Garland, D. and Perry, D. (1995). Special issue on software architectures. *IEEE Transactions on Software Engineering.*

[Garland et al., 1993] Garland, S. J., Guttag, J. V., and Horning, J. J. (1993). An overview of Larch. In *Functional Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 329–348. Springer-Verlag.

[Gaspari et al., 1998] Gaspari, M., Motta, E., and Fensel, D. (1998). Exploiting automated theorem proving in UPML: Automatic configuration of PSM from PSMs libraries. Technical report, Robotics Institute, Carnegie Mellon University.

[Gaspari et al., 1999] Gaspari, M., Motta, E., and Fensel, D. (1999). Automatic selection of problem solving libraries based on competence matching. In *ECOOP Workshops*, pages 10–11.

[Gasser and Briot, 1992] Gasser, L. and Briot, J.-P. (1992). Object-based concurrent programming and distributed artificial intelligence. In Avouris, N. M. and Gasser, L., editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 81–107. Kluwer, Dordrecht.

[Genesereth et al., 1997] Genesereth, M., Keller, A., and Duschka, O. (1997). Infomaster: an information integration system. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 539–542.

[Genesereth and Ketchpel, 1997] Genesereth, M. R. and Ketchpel, S. P. (1997). Software agents. *Communications of the ACM*, 37(7).

[Gennari and Tu, 1994] Gennari, J. and Tu, S. (1994). Mapping domains to methods in support of reuse. *International Journal on Human Computer Studies*, 41:399–42.

[Georgeff, 1983] Georgeff, M. (1983). Communication and interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence*.

[Georgeff and Lansky, 1987] Georgeff, M. and Lansky, A. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, volume 2, pages 677–682.

[Giampapa and Sycara, 2001] Giampapa, J. A. and Sycara, K. (2001). Conversational case-based planning for agent team coordination. *Lecture Notes in Computer Science*, 2080.

[Giampapa and Sycara, 2002] Giampapa, J. A. and Sycara, K. (2002). Team-oriented agent coordination in the retsina multi-agent system. Technical Report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. Presented at AAMAS 2002 Workshop on Teamwork and Coalition Formation.

[Giunchiglia and Shvaiko, 2003] Giunchiglia, F. and Shvaiko, P. (2003). Semantic matching. Technical Report DIT-03-013, Informatica e Telecomunicazioni, University of Trento.

[Glaser, 1996] Glaser, N. (1996). *Contribution to Knowledge Modelling in a Multi-Agent Framework*. PhD thesis, L'Université Henri Poincaré, Nancy I, France.

[Goguen et al., 1996] Goguen, J., Nguyen, D., Meseguer, J., Luqi, Zhang, D., and Berzins, V. (1996). Software component search. journal of systems integration. *Journal of Systems Integration,*, 6:93–134.

[Gómez and Benjamins, 1999] Gómez, A. and Benjamins, V. (1999). Overview of knowledge sharing and reuse of components: Ontologies and problem-solving methods. In *Proceedingsof Workshop on Ontologies and Problem-Solving Methods of IJCAI*.

[Gómez and Abasolo, 2002] Gómez, M. and Abasolo, C. (2002). Improving meta-search by using query-weighting and numerical aggregation operators. In *Proceedings 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*.

[Gómez et al., 2001] Gómez, M., Abasolo, C., and Plaza, E. (2001). Domain-independent ontologies for cooperative information agents. In *Proceedings*

of the Fifth International Workshop on Cooperative Information Agents, volume 2128 of *Lecture Notes in Artificial Intelligence*, pages 118–129. Springer-Verlag.

[Gómez et al., 2002] Gómez, M., Abasolo, C., and Plaza, E. (2002). Problem-solving methods and cooperative information agents. *International Journal on Cooperative Information Systems*, 11(3-4):329–354.

[Gómez and Abasolo, 2003] Gómez, M. and Abasolo, J. M. (2003). A general framework for meta-search based on query weighting and numerical aggregation operators. In *Intelligent Systems for Information Processing: From Representation to Applications*, pages 129–140. Elsevier Science.

[Gómez et al., 2003a] Gómez, M., Abasolo, J. M., and Plaza, E. (2003a). Description and configuration of multi-agent systems at the knowledge level. In Aguiló, I., Valverde, L., and Escrig, M., editors, *Artificial Intelligence Research and Development*, volume 100 of *Frontiers in Artifical Intelligence and Applications*, pages 221–232. IOS Press.

[Gómez et al., 2003b] Gómez, M., Abasolo, J. M., and Plaza, E. (2003b). Orcas: Open, reusable and configurable multi-agent systems. Third price in the Infrastructures Category.

[Gómez and Plaza, 2004a] Gómez, M. and Plaza, E. (2004a). Extending matchmaking to maximize capability reuse (to appear). In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*.

[Gómez and Plaza, 2004b] Gómez, M. and Plaza, E. (2004b). A knowledge based framework for configuring agent teams on-demand. In *Proceedings of the Workshop on Semantic Intelligent Middleware for Interoperable Systems*.

[Grosz et al., 1999] Grosz, B. J., Hunsberger, L., and Kraus, S. (1999). Planning and acting together. *The AI Magazine*, 20(4):23–34.

[Grosz and Kraus, 1993] Grosz, B. J. and Kraus, S. (1993). Collaborative plans for group activities. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 367–373.

[Grosz and Kraus, 1996] Grosz, B. J. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357.

[Grosz and Sidner, 1990] Grosz, B. J. and Sidner, C. L. (1990). Plans for discourse. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA.

[Gruber, 1993a] Gruber, T. R. (1993a). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N. and Poli, R., editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands. Kluwer Academic Publishers.

[Gruber, 1993b] Gruber, T. R. (1993b). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

[Guarino, 1997a] Guarino, N. (1997a). Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In Pazienza, M., editor, *Summer School on Information Extraction*, pages 139–170. Springer Verlag.

[Guarino, 1997b] Guarino, N. (1997b). Understanding, building, and using ontologies: A commentary to using explicit ontologies in kbs development, by van heijst, schreiber, and wielinga. *International Journal of Human and Computer Studies*, 46:293–310.

[Hall, 1993] Hall, R. J. (1993). Generalized behavior-based retrieval. In *Proceedings of 15th International Conference on Software Engineering*, pages 371–380.

[Han, 1999] Han, J. (1999). Semantic and usage packaging for software components. In *Workshops of 13th European Conference on Object-Oriented Programming*, pages 7–8.

[Haustein and Ludecke, 2000] Haustein, S. and Ludecke, S. (2000). Towards information agent interoperability. In Klusch, M. and Kerschberg, L., editors, *Proceedings Cooperative Information Agents*, Lecture Notes in Computer Science, pages 208–219. Springer.

[Herlea et al., 1999] Herlea, D. E., Jonker, C. M., Treur, J., and Wijngaards, N. J. E. (1999). Specification of behavioural requirements within compositional multi-agent system design. In Garijo, F. J. and Boman, M., editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, pages 8–27. Springer-Verlag: Heidelberg, Germany.

[Horrocks, 2002] Horrocks, I. (2002). DAML+OIL: A reasonable web ontology language. In *Proceedings the 7th Conference on Extending Database Technology*, volume 2287 of *Lecture Notes in Computer Science*, pages 2–13. Springer-Verlag.

[Huguet et al., 2002] Huguet, M.-P., Esteva, M., Parsons, S., Sierra, C., and Wooldridge, M. (2002). Model checking electronic institutions. In *Proceedings of the ECAI Workshop on Model Checking Artificial Intelligence*.

[Iglesias et al., 1998] Iglesias, C., Garrijo, M., and Gonzalez, J. (1998). A survey of agent-oriented methodologies. In *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages*, volume 1555 of *Lecture Notes in Artificial Intelligence*, pages 317–330.

[Iglesias et al., 1997] Iglesias, C. A., Garijo, M., Centeno-Gonzalez, J., and Velasco, J. R. (1997). Analysis and design of multiagent systems using MAS-common KADS. In *Agent Theories, Architectures, and Languages*, pages 313–327.

[IIIA, 2003] IIIA (2003). Dialogical institutions. http://e-institutor.iiia.csic.es/.

[International Foundation on Cooperative Information Systems, 1994] International Foundation on Cooperative Information Systems (1994). Second international conference on cooperative information systems.

[Iribarne et al., 2002] Iribarne, L., Troya, J., and Vallecillo, A. (2002). Selecting software components with multiple interfaces. In *Proceeding of the 28th EUROMICRO Conference – Component-Based Software Engineering*, pages 26–32.

[Jennings, 1993] Jennings, N. R. (1993). Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250.

[Jennings, 2000] Jennings, N. R. (2000). On-agent-based software engineering. *Artificial Intelligence*, 117:227–296.

[Jennings and Campos, 1997] Jennings, N. R. and Campos, J. (1997). Towards a social level characterisation of socially responsible agents. *IEEE Proc. Software Engineering*, 144(1):11–25.

[Jennings et al., 1992] Jennings, N. R., Mamdani, E., Laresgoiti, I., Perez, J., and Corera, J. (1992). GRATE: a general framework for cooperative problem solving. *IEEE-BCS Journal of Intelligent Systems Engineering*, 1(2):102–104.

[Jennings et al., 1998] Jennings, N. R., Sycara, K., and Woolridge, M. (1998). A roadmap of agent-research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):227–296.

[Jeusfeld and Papazoglou, 1996] Jeusfeld, M. A. and Papazoglou, M. (1996). Information brokering: Design, search and transformation. *Aachener Informatik-Bericht*, 96(18).

[John H. Gennari and Musen, 1998] John H. Gennari, W. G. and Musen, M. (1998). A method-description language: an initial ontology with examples. In *Proceedings 11th Workshop on Knowledge Acquisition, Modelling and Management*.

[Jonathan Lee and Chiang, 2002] Jonathan Lee, K. F. L. and Chiang, W. (2002). Possibilistic petri nets as a basis for agent service description language. Submitted to Fuzzy Sets and Systems.

[Juhasz and Paul, 2002] Juhasz, Z. and Paul, P. (2002). Scalability analysis of the contract net protocol. In *Proceedings of the 2nd International Workshop on Agent based Cluster and Grid Computing at IEEE International Symposium on Cluster Computing and the Grid*, Berlin, Germany.

[Kendall et al., 1995] Kendall, E. A., Malkoun, M. T., and Jiang, C. H. (1995). A methodology for developing agent based systems for enterprise integration. In *First Australian Workshop on Distributed Artificial Intelligence*.

[Kiepuszewski, 2002] Kiepuszewski, B. (2002). *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia.

[Kinny and Georgeff, 1996] Kinny, D. and Georgeff, M. (1996). Modelling and design of multi-agent systems. In *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Lecture Notes in Artificial Intelligence.

[Kinny et al., 1992] Kinny, D., Ljungberg, M., Rao, A. S., Sonenberg, E., Tidhar, G., and Werner, E. (1992). Planned team activity. In Castelfranchi, C. and Werner, E., editors, *Artificial Social Systems. Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 830 of *Lecture Notes in Artificial Intelligence*, pages 226–256. Springer-Verlag: Heidelberg, Germany.

[Kirk et al., 1995] Kirk, T., Levy, A. Y., Sagiv, Y., and Srivastava, D. (1995). The Information Manifold. In Knoblock, C. and Levy, A., editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California.

[Klein, 2000] Klein, M. (2000). The Challenge: Enabling Robust Open Multi-Agent Systems.

[Klinker et al., 1991] Klinker, G., Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition*, 3:117–135.

[Knoblock et al., 1994] Knoblock, C. A., Arens, Y., and Hsu, C.-N. (1994). Cooperating agents for information retrieval. In *Proceedings of the 2nd International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada. University of Toronto Press.

[Labrou and Finin, 1997] Labrou, Y. and Finin, T. (1997). Semantics and conversations for an agent communication language. In Pollack, M. E., editor, *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence*, pages 584–591, Nagoya, Japan. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Larson and Sandholm, 2000] Larson, K. and Sandholm, T. (2000). Anytime coalition structure generation: An average case study. *Journal of Experimental and Theoretical AI*, 11:1–20.

[Lemahieu, 2001] Lemahieu, W. (2001). Web service description, advertising and discovery: Wsdl and beyond. In Vandenbulcke, J. and Snoeck, M., editors, *New Directions In Software Engineering*. Leuven University Press.

[Lesser, 1991] Lesser, V. R. (1991). A retrospective view of distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Distributed Artificial Intelligence*, 1(1):63–83.

[Lesser et al., 1989] Lesser, V. R., Durfee, E., and Corkill, D. (1989). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83.

[Levesque, 1990] Levesque, H. J. (1990). On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 94–99.

[Levy et al., 1996] Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996). Query-answering algorithms for information agents. In *Proceedings of the AAAI*, pages 40–47.

[Liskov and Wing, 1993] Liskov, B. and Wing, J. M. (1993). A new definition of the subtype relation. In Nierstrasz, O. M., editor, *Proceedings 7th European Conference on Object-Oriented Programming*, volume 707 of *Lecture Notes in Computer Science*, pages 118–141. Springer-Verlag.

[Luck et al., 1997] Luck, M., Griffiths, N., and d'Inverno, M. (1997). From agent theory to agent construction: A case study. In Müller, J. P., Wooldridge, M. J., and Jennings, N. R., editors, *Proceedings of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III*, volume 1193, pages 49–64. Springer-Verlag: Heidelberg, Germany.

[Luck et al., 2003] Luck, M., McBurney, P., and Preist, C. (2003). *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink.

[Maarek et al., 1991] Maarek, Y., Berry, D., and Kaiser, G. (1991). An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17(8):800–813.

[Martin et al., 1999] Martin, D. L., Cheyer, A. J., and Moran, D. B. (1999). The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128. OAA.

[Martin et al., 1997] Martin, D. L., Oohama, H., Moran, D., and Cheyer, A. (1997). Information brokering in an agent architecture. In *Proceedings of the 2nd International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*.

[McDermott, 1988] McDermott, J. (1988). Toward a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–256. Kluwer Academic.

[McIlraith and Son, 2001] McIlraith, S. and Son, T. C. (2001). Adapting golog for programming the semantic web. In *Working Notes of the 5th International Symposium on Logical Formalizations of Commonsense Reasoning*.

[McIlraith et al., 2001] McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems (Special Issue on the Semantic Web)*, 16(2):46–53.

[Mikhajlova, 1999] Mikhajlova, A. (1999). *Ensuring Correctness of Object and Component Systems*. PhD thesis, Abo Akademi University.

[Mili et al., 1997] Mili, A., Mili, R., and Mittermeir, R. (1997). Storing and retrieving software components: A refinement based system. *IEEE Transactions on Software Engineering*, 23(7):445–460.

[Mili et al., 1995] Mili, H., Mili, F., and Mili, A. (1995). Reusing software: Issues and research directions. *Software Engineering*, 21(6):528–562.

[Monica Crubezy and Musen, 2001] Monica Crubezy, W. L. and Musen, M. A. (2001). The internet reasoning service: Delivering configurable problem-solving components to web users.

[Motta, 1999] Motta, E. (1999). *Reusable Components for Knowledge Modelling*, volume 53 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

[Motta et al., 1999] Motta, E., Fensel, D., Gaspari, M., and Benjamins, A. (1999). Specifications of knowledge components for reuse. In *Proceedings of SEKE '99, 1999.*

[Moulin and Brassard, 1996] Moulin, B. and Brassard, M. (1996). A scenario-based design method and an environment for the development of multiagent systems. In *Proceedings of the 1th Australian Workshop on Distributed Artificial Intelligence*, volume 1087 of *Lecture Notes in Artificial Intelligence*, pages 216–231.

[Müller, 1996] Müller, H. J. (1996). Towards agent systems engineering. *International Journal on Data and Knowledge Engineering. Special Issue on Distributed Expertise*, 23:217–245.

[Munoz-Avila et al., 1999] Munoz-Avila, H., Aha, D. W., Breslow, L., and Nau, D. S. (1999). HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 879–885. AAAI Press.

[Musen, 1998] Musen, M. (1998). Modern architectures for intelligent systems: Reusable ontologies and problem-solving methods. In *Proceedings of AMIA Fall Symposium.*

[Narayan and McIlraith, 2002] Narayan, S. and McIlraith, S. (2002). Simulation, verification and automated composition of web services. In *Proceedings of the 11th International World Wide Web Conference*, pages 77–88. ACM Press.

[Narendra, 2003] Narendra, N. (2003). An agent-oriented architectural framework for enacting and managing web services. Agenticies Working Group on Service Description and Composition.

[Newell, 1982] Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 28(2):87–127.

[Nii, 1989] Nii, H. (1989). Blackboard systems. In Bar, A., P.R.Cohen, and Feigenbaum, E., editors, *The Handbook of Artificial Intelligence IV*, pages 1–82. Addison-Wesley.

[Nodine et al., 1999] Nodine, M., Bohrer, W., and Ngu, A. (1999). Semantic brokering over dynamic heterogeneous data sources in infosleuth. In *ICDE*, pages 358–365.

[Nodine and Unruh, 1999] Nodine, M. and Unruh, A. (1999). Constructing robust conversation policies in dynamic agent communities. In *Proceedings of the AGENTS'99 Workshop on Specifying and Implementing Conversation Policies*.

[Noriega, 1997] Noriega, P. (1997). *Agent-Mediated Auctions: The Fish-Market Metaphor*. PhD thesis, Universitat Autnoma de Barcelona.

[Norman, 1994] Norman, T. (1994). Motivated goal and action selection. In *Working Notes of the AISB worskhop, Models or Behaviours, which way forward for robotics?*

[Nwana and Woolridge, 1996] Nwana, H. S. and Woolridge, M. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244.

[O'Hare and Woolridge, 1992] O'Hare, G. and Woolridge, M. (1992). A software engineering perspective on multi-agent system design. In Avouris, N. M. and Gasser, L., editors, *Distributed Artificial Intelligence: Theory and Practice*, pages 109–127. Kluwer Academic Publishers.

[Orsvarn, 1996] Orsvarn, K. (1996). Principles for libraries of task decomposition methods – conclusions from a case-study. In *Proceedings of 9th European Knowledge Acquisition Workshop*, volume 1076 of *Lecture Notes in Artificial Intelligence*, pages 48–65. Springer-Verlag.

[Ousterhout, 1990] Ousterhout, J. K. (1990). Tcl: An embeddable command language. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 133–146, Berkeley, CA. USENIX Association.

[Panzarasa and Jennings, 2001] Panzarasa, P. and Jennings, N. R. (2001). The organisation of sociality: A manifesto for a new science of multiagent systems. In *In Proceedings of the Tenth European Workshop on Multi-Agent Systems*.

[Paolucci et al., 2002] Paolucci, M., Kawmura, T., Payne, T., and Sycara, K. (2002). Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference*.

[Park et al., 1998] Park, J. Y., Gennari, J. H., and Musen, M. A. (1998). Mappings for reuse in knowledge-based systems. In *Proceedings 11th Workshop on Knowledge Acquisition, Modelling and Management*.

[Payne et al., 2001] Payne, T. R., Paolucci, M., and Sycara, K. (2001). Advertising and matching daml-s service descriptions. In *Semantic Web Working Symposium (SWWS)*.

[Pednault, 1989] Pednault, E. P. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. Morgan Kaufmann.

[Peltz, 2003] Peltz, C. (2003). Web services orchestration. a review of emerging technologies, tools and standards. Technical report, Hewlett Packard, Co.

[Penix, 1998] Penix, J. (1998). *Automated Component Retrieval and Adaptation Using Formal Specifications*. PhD thesis, University of Cincinnati.

[Penix and Alexander, 1997] Penix, J. and Alexander, P. (1997). Component reuse and adaptation at the specification level. In *Proceedings of the 8th Annual Workshop on Institutionalizing Software Reuse*.

[Penix and Alexander, 1999] Penix, J. and Alexander, P. (1999). Efficient specification-based component retrieval. *Automated Software Engineering: An International Journal*, 6(2):139–170.

[Penix et al., 1995] Penix, J., Baraona, P., and Alexander, P. (1995). Classification and retrieval of reusable components using semantic features. In *Proceedings of the 10th Knowledge-Based Software Engineering Conference*, pages 131–138.

[Piresa et al., 2003] Piresa, P. F., Benevides, M. R. F., and Mattoso:, M. (2003). Building reliable web services compositions. In *LNCS 2593*, volume 2593 of *Lecture Notes in Computer Science*, pages 59–72. Springer-Verlag.

[Plaza and Arcos, 2002] Plaza, E. and Arcos, J. L. (2002). Constructive adaptation. In Craw, S. and Preece, A., editors, *Advances in Case-Based Reasoning. Proceedings 6th ECCBR*, volume 2416 of *Lecture Notes in Artificial Intelligence*, pages 306–320.

[Poek and Gappa, 1993] Poek, K. and Gappa, U. (1993). Making role-limiting shells more flexible. In Aussenac, N. et al., editors, *Knowledge Acquisition for Knowledge-Based Systems. Proceedings of EKAW'93*.

[Prieto-Daz, 1987] Prieto-Daz, R. (1987). Classifying software for reusability. *IEEE Software*, 4(1).

[Puerta et al., 1992] Puerta, A., Egar, J., Tu, S. W., and Musen, M. A. (1992). A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition*, 4:171–196.

[Pynadath et al., 1999] Pynadath, D. V., Tambe, M., Chauvat, N., and Cavedon, L. (1999). Toward team-oriented programming. In *Agent Theories, Architectures, and Languages*, pages 233–247.

[Rao, 1994] Rao, A. S. (1994). Means-end plan recognition : Towards a theory of reactive recognition. In Torasso, P. and Jon Doyle, a. E. S., editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 497–508. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Rao and Georgeff, 1991] Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P. (1995). BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco.

[Rao et al., 1992] Rao, A. S., Georgeff, M. P., and Sonenberg, E. A. (1992). Social plans: A preliminary report. In Werner, E. and Demazeau, Y., editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76, Kaiserslautern, Germany. Elsevier Science B.V.: Amsterdam, Netherland.

[Rising, 2000] Rising, L. (2000). *The Pattern Almanac 2000*. Software PAtterns Series. Addison Wesley.

[Rodríguez-Aguilar, 1997] Rodríguez-Aguilar, J. A. (1997). *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Universitat Autnoma de Barcelona.

[Rollins and Wing, 1991] Rollins, E. J. and Wing, J. M. (1991). Specifications as search keys for software libraries. In Furukawa, K., editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 173–187, Paris, France. The MIT Press.

[Rosario and Ibrahim, 1994] Rosario, M. and Ibrahim, G. (1994). A similarity measure for retrieving software artifacts. In *Proceedings of the Sixth International Conference on Software Engineering and Knowledge Engineering*, pages 478–485.

[Sandholm et al., 1998] Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohme, F. (1998). Anytime coalition structure generation with worst case guarantees. In *Proceedings AAAI Innovative Applications of Artificial Intelligence*, pages 46–53.

[Sandholm, 1993] Sandholm, T. W. (1993). An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 295–308, Hidden Valley, Pennsylvania.

[Schreiber et al., 1994a] Schreiber, A., Wielinga, B. J., Ackermans, J., Van de Velde, W., and Hoog, R. D. (1994a). CommonKADS: A comprehensive methodology for kbs development. *IEEE Expert*, 9(6):28–37.

[Schreiber et al., 1994b] Schreiber, G., Wielinga, B., Akkermans, H., Van de Velde, W., and Anjewierden, A. (1994b). CML: The CommonKADS conceptual modelling language. In Steels, L., Schreiber, G., and Van de Velde, W., editors, *A Future for Knowledge Acquisition: Proceedings of EKAW'94*, pages 1–25, Berlin, Heidelberg. Springer.

[Schreiber et al., 1993] Schreiber, G., Wielinga, B. J., and Breuker, J. A. (1993). *KADS: a Principled Approach to Knowledge-Based System Development*, volume 11 of *Knowledge-Based Systems*. Academic Press.

[Searle, 1969] Searle, J. (1969). *Speech Acts: An Essay in the Philosophy of Language*. PhD thesis, Cambridge University Press.

[Shadbolt et al., 1993] Shadbolt, N., Motta, E., and Rouge, A. (1993). Constructing knowledge-based systems. *IEEE Software*, 10(6):34–39.

[Shaw and Garlan, 1996] Shaw, M. and Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

[Shehory and Kraus, 1998] Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200.

[Shehory et al., 1997] Shehory, O., Sycara, S. K., and Jha, S. (1997). Multi-agent coordination through coalition formation. In *Intelligent Agents IV: Agent Theories, Architectures and Languages*, volume 1365, pages 143–154. Springer.

[Singh, 1994] Singh, M. P. (1994). *Multiagent Systems: A theoretical framework for intentions, know-how and communication*, volume 799 of *Lecture Notes in Artificial Intelligence*. Springer Verlag.

[Singh, 1998] Singh, M. P. (1998). The intentions of teams: Team structure, endodeixis, and exodeixis.

[Singh et al., 1993] Singh, M. P., Huhns, M. N., and M.Stephens, L. (1993). Declarative representations of multi-agent systems. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):721–739.

[Smith, 1994] Smith, G. (1994). Formal definitions of behavioural compatibility for active and passive objects. In *Proceedings 1st Asia-Pacific Software Engineering Conference*, pages 336–344. IEEE Computer Society Press.

[Smith, 1940] Smith, R. G. (1940). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113.

[Sonenberg et al., 1994] Sonenberg, E., Tidhar, G., Werner, E., Kinny, D., Ljungberg, M., and Rao, A. (1994). Planned team activity. In *Articial Social Systems*, Lecture Notes in Articial Intelligence, pages 227–256.

[Splunter et al., 2003] Splunter, S. V., Wijngaards, N. J., and Brazier, F. M. (2003). Structuring agents for adaptation. In Alonso, E., D., K., and Kazakov, D., editors, *Adaptive Agents and Multi-Agent Systems*, Lecture Notes in Artificial Intelligence (LNAI) 2636. Springer-Verlag Berlin.

[Stab et al., 2003] Stab, S. et al. (2003). Web service: Been there, done what. *IEEE Intelligent Systems.*, 18(1).

[Stader and Macintosh, 1999] Stader, J. and Macintosh, A. (1999). Capability modelling and knowledge management. In *Applications and Innovations in Expert Systems VII*, pages 33–50. Springer Verlag.

[Steels, 1988] Steels, L. (1988). The deepening of expert systems. *AI Communications*, 1(1):9–17.

[Steels, 1990] Steels, L. (1990). Components of expertise. *AI Magazine*, 11(2):28–49.

[Steels, 1993] Steels, L. (1993). The componential framework and its role in reusability. In David, J., Krivine, J., and Simmons, R., editors, *Second generation expert systems*, pages 273–298. Springer Verlag.

[Studer et al., 1998] Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197.

[Studer et al., 1996] Studer, R., Eriksson, H., Gennari, J., Tu, S., Fensel, D., and Musen, M. (1996). Ontologies and the configuration of problem-solving methods. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop.*

[Sycara et al., 1996] Sycara, K. P., Decker, K., Pannu, A., Williamson, M., and Zeng, D. (1996). Distributed intelligent agents. *IEEE Expert*, 11(6):36–46.

[Sycara et al., 1999a] Sycara, K. P., Klusch, M., Widoff, S., and Lu, J. (1999a). Dynamic service matchmaking among agents in open information environments. *SIGMOD*, 28(1):47–53.

[Sycara et al., 1999b] Sycara, K. P., Lu, J., Klusch, M., and Widoff, S. (1999b). Matchmaking among heterogeneous agents on the internet. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace.*

[Sycara et al., 2001] Sycara, K. P., Paolucci, M., Velsen, M. V., and Giampapa, J. A. (2001). The RETSINA MAS infrastructure. Technical report, Robotics Institute, Carnegie Mellon University.

[Sycara et al., 2002] Sycara, K. P., Widoff, S., Klusch, M., and Lu, J. (2002). Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5:173–203.

[Szyperski, 1996] Szyperski, C. (1996). Independently extensible systems – software engineering potential and challenges. In *Proceedings of the 19th Australian Computer Science Conference*, Melbourne, Australia.

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124.

[Tambe et al., 1999] Tambe, M., Adibi, J., Alonaizon, Y., Erdem, A., Kaminka, G. A., Marsella, S., and Muslea, I. (1999). Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110(2):215–239.

[Tambe et al., 2000] Tambe, M., Pynadath, D. V., Chauvat, N., Das, A., and Kaminka, G. A. (2000). Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the International Conference on Multiagent Systems*, pages 301–308, Boston, MA.

[Tate, 1998] Tate, A. (1998). Roots of SPAR- shared planning and activity representation. *The Knowledge Engineering Review, Special Issue on Putting Ontologies to Use*, 13(1):121–128.

[Terpstra et al., 1993] Terpstra, P., van Heijst, G., Wielinga, B., and Shadtbolt, N. (1993). Knowledge acquisition support through generalised directive models. In David, J., Krivine, J., and Simmons, R., editors, *Second Generation Expert Systems*, pages 428–455. Springer-Verlag.

[The DAML-S Consortium, 2001] The DAML-S Consortium (2001). Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop*.

[Tidhar et al., 1996] Tidhar, G., Rao, A., and Sonenberg, E. (1996). Guided team selection. In *In Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*.

[Tidwell, 2000] Tidwell, D. (2000). Web services. the web's next revolution. http://www-106.ibm.com/developerworks/webservices.

[Torra, 1996] Torra, V. (1996). Weighted owa operators for synthesis of information. In *Proceedings 5th IEEE Inter. Conference on Fuzzy Systems*, pages 966–971.

[Tu et al., 1995] Tu, S. W., Eriksson, H., Gennari, J. H., Shahar, Y., and Musen, M. A. (1995). Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools: application of PROTEGE-II to protocol-based decision support. *Artificial Intelligence in Medicine*, 7(3):257–289.

[Valente and Lockenhoff, 1993] Valente, A. and Lockenhoff, C. (1993). Organization as guidance: A library of assessment models. In *Proceedings of the Seventh European Knowledge Acquisition Workshop*, volume 723 of *Lecture Notes in Artificial Intelligence*.

[Valente et al., 1994] Valente, A., Van de Velde, W., and Breuker, J. (1994). The commonkads expertise modelling library. In Breuker, J. and Van de Velde, W., editors, *CommonKADS Library for Expertise Modeling*, volume 21 of *Frontiers in Artificial Intelligence and Applications*, pages 31–56. IOS-Press.

[Van de Velde, 1993] Van de Velde, W. (1993). Issues in knowledge level modelling. In David, J., Krivine, J., and Simmons, R., editors, *Second generation expert systems*, pages 211–231. Springer Verlag.

[van der Aalst and ter Hofstede, 2002] van der Aalst, W. and ter Hofstede, A. (2002). Yawl: Yet another workflow language. QUT Technical report FIT-TR-2002-06, Queensland University of Technology.

[van der Aalst et al., 2001] van der Aalst, W., Verbeek, H., and Kumar, A. (2001). Rl/woflan: Verification of an xml/petri-net based language for inter-organizational workflows. In Altinkemer, K. and Chari, K., editors, *Proceedings of the 6th Informs Conference on Information Systems and Technology*, pages 30–45.

[van Heijst, 1995] van Heijst, G. (1995). *The Role of Ontologies in Knowledge Engineering*. PhD thesis, University of Amsterdam.

[Vasconcelos et al., 2001] Vasconcelos, W. W., Sabater, J., Sierra, C., and Querol, J. (2001). Skeleton-based agent development for electronic institutions. In *Proceedings UKMAS*.

[Vaughan-Nichols, 2002] Vaughan-Nichols, S. J. (2002). Web services: Beyond the hype. *Computer*, pages 18–21.

[Vidal and Durfee, 1995] Vidal, J. M. and Durfee, E. H. (1995). Task planning agents in the UMDL. In *Proceedings of the Fourth International Conference on Information and Knowledge Management (CIKM) Workshop on Intelligent Information Agents*.

[Vidal et al., 1998] Vidal, J. M., Mullen, T., Weinstein, P., and Durfee, E. H. (1998). The UMDL service market society. In *Proceedings of the Second International Conference on Autonomous Agents*.

[Walther et al., 1992] Walther, E., Eriksson, H., and Musen, M. (1992). Plug and play: Construction of task specific expert-system shells using sharable context ontologies. Technical Report KSl-92-40, Knowledge Systems Laboratory, Stanford University.

[Walton and Krabbe, 1995] Walton, D. N. and Krabbe, E. C. (1995). *Commitment in Dialogue*. SUNNY Press.

[Wegner, 1984] Wegner, P. (1984). Capital-intensive software technology. *IEEE Software*, 1(3):7–45.

[White and Sleeman, 1999] White, S. and Sleeman, D. H. (1999). A constraint-based approach to the description of competence. In *Proceedings of Knowledge Acquisition, Modeling and Management*, volume 1621 of *Lecture Notes in Artificial Intelligence*, pages 291–308. Springer Verlag.

[Wickler and Tate, 1999] Wickler, G. and Tate, A. (1999). Capability representations for brokering: A survey. submitted to the knowledge engineering review.

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*.

[Wiederhold, 1993] Wiederhold, G. (1993). Intelligent integration of information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 434–437.

[Wiederhold and Genesereth, 1997] Wiederhold, G. and Genesereth, M. R. (1997). The conceptual basis for mediation services. *IEEE Expert*, 12(5):38–47.

[Wielinga et al., 1993] Wielinga, B. J., Schreiber, A. T., and Breuker, J. A. (1993). Kads: A modelling approach to knowledge engineering. In Buchanan, B. G. and Wilkins, D. C., editors, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, pages 92–116. Kaufmann, San Mateo, CA.

[Wil M. P. van der Aalst, 2002] Wil M. P. van der Aalst, K. M. v. H. (2002). *Workflow Management: Models, Methods, and Systems*. MIT Press.

[Wilsker, 1996] Wilsker, B. (1996). Study of multi-agent collaboration theories. Technical report, University of Southern California. Information Science Institute. RS-96-449.

[Wittig et al., 1994] Wittig, T., Jennings, N., and Mamdani, E. (1994). ARCHON - a framework for intelligent cooperation. *IEE-BCS Journal of Intelligent Systems Engineering*, 3(3):168–179.

[Wong and Sycara, 2000] Wong, H. C. and Sycara, K. (2000). A taxonomy of middle-agents for the internet. In *Proceedings of the International Conference on Multi-Agent Systems*.

[Wooldridge, 1998] Wooldridge, M. (1998). Agents and software engineering. *AI*IA Notizie*, XI(3):31–37.

[Wooldridge and Jennings, 1994] Wooldridge, M. and Jennings, N. R. (1994). Towards a theory of cooperative problem solving. In *Proceedings Modelling Autonomous Agents in a Multi-Agent World*, pages 15–26.

[Wooldridge and Jennings, 1999] Wooldridge, M. and Jennings, N. R. (1999). The cooperative problem-solving process. *Journal of Logic and Computation*, 9(4):563–592.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312.

[Xu and Shatz, 2001] Xu, H. and Shatz, S. M. (2001). A framework for modeling Agent-Oriented software. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 57–64.

[Yellin and Strom, 1997] Yellin, D. M. and Strom, R. E. (1997). Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333.

[Yoav Shoham, 1993] Yoav Shoham (1993). Agent-oriented programming. *Artificial Intelligence*, (60):51–92.

[Zaremski and Wing, 1995] Zaremski, A. M. and Wing, J. M. (1995). Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170.

[Zaremski and Wing, 1997] Zaremski, A. M. and Wing, J. M. (1997). Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369.

[Zhang, 2000] Zhang, Z. (2000). Enhancing component reuse using search techniques. In *Proceedings of 23rd conference on Information System Research in Scandinavia*.