VNiVERSiTY OF SALAMANCA

DEPARTMENT OF COMPUTER SCIENCE AND AUTOMATION
SCIENCES FACULTY

VNiVERSiDAD
D SALAMANCA

PHD THESIS

ORGANIZATION BASED MULTIAGENT ARCHITECTURE FOR DISTRIBUTED ENVIRONMENTS

AVTHOR
Aitor Mata Conde

SVPERVISORS
Dr. Belén Pérez Lancho
Dr. Emilio S. Corchado Rodríguez

~ MARCH, 2010 ~
Salamanca, Spain

The present PhD. thesis document entitled “Organization Based Multiagent Architecture for Distributed Environments” submitted by Mr. Aitor Mata Conde to the Department of Computer Science and Automation of the University of Salamanca in partial fulfilment of the requirements for the degree of Doctor in Computer Science and Automatics has been carried out under the supervision of Dr. Belén Pérez Lancho, Senior Lecturer at the Department of Computer Science and Automation of the University of Salamanca, and Dr. Emilio S. Corchado Rodríguez, Senior Lecturer at the Department of Computer Science and Automation of the University of Salamanca

Salamanca, march 19th, 2010.

The Supervisors,

The graduate

Signed, Dr. Belén Pérez-Lancho
Senior Lecturer
University of Salamanca, Spain

Signed, Mr. Aitor Mata Conde

Signed, Dr. Emilio S. Corchado
Senior Lecturer
University of Salamanca, Spain

”We can only see a short distance ahead,

but we can see plenty there that needs to be done. “

A. M. Turing

“Computing machinery and intelligence”

Acknowledgements

Finishing a work like this takes a lot of time and effort and I have to thank many people. Despite any attempt to thank everyone involved in the development of this investigation, please forgive any unintentional omission.

I want to begin acknowledging my two supervisors, Belén and Emilio, who have been a great help especially at the end of this complex process, their perspective and compatible but different point of views have made it easier. I also want to thank Professor Juan for his invaluable help from the beginning and all through the process of the development of this PhD. thesis.

I have to remember Deanna here, for helping me with all that English expressions and corrections in this document.

I also have to cite here Dante and Javi, for their help and advices to easier face all the process and Fran and Sara for being helpful partners, sharing problems, congresses and times.

Alfonso de Maruri must also be acknowledged, for being a great help with the internal guidelines of an investigation of this kind, and for being a constant help with the guidelines of my own life.

My parents and brother have been a helpful hand and table for

receiving and listening to me when tons of work accumulated over my shoulders.

My family in law has also been there, specially my nephews that have created a new expression: when somebody is concentrated doing anything or fixedly regarding something, he or she is *making the thesis.*

Miro, my father in law, who left us during this process. His presence has been an important companion in some difficult moments.

It is fair to remember here all the members of the Department of Computer Science and Automation of the University of Salamanca. They gave me, from the very beginning of my university studies, a complete spectre of this science with its lights and shadows, but always exciting.

I also want to thank the company of a set of authors that have made the fulfilment of this thesis easier. They have put light in the dark moments, introducing external elements that have enriched the process. They are, in almost chronologically order: Jaime Bayly [Bayly, 2005], Josep María Gironella [Gironella, 1952], Frank McCourt [McCourt, 2006], Javier Reverte [Reverte, 2006], María Luisa Prada [Prada, 2004], Ken Follett [Follett, 2007], Muriel Barbery [Barbery, 2007], Jean Auel [Auel, 2005], Isabel Allende, twice [Allende, 2005, , 2003], Marta Rivera de la Cruz [Rivera de la Cruz, 2009], Fernando Savater [Savater, 2008], Stieg Larsson, three times [Larsson, 2009b, , 2008, , 2009a], Mathias Malzieu, [Malzieu, 2009], Fernando Sánchez Dragó [Sánchez Dragó, 2009] and Lao Tse[Tse, 2007]. All these authors and books have served as a varied counterpoint to all the information, data, experiments, and results showed through this document. It is also necessary to cite Silvio here, to make me remind teenage rhythms and Joel Fleischman, Chris Stevens and all their neighbours, to show me again that other type of daily life is possible.

And last but, of course, not least I want to specially thank Yolanda, for being always there, her help and presence have made this possible.

Abstract

Distributed environments represent a complex field in which applied solutions should be flexible and include significant adaptation capabilities. These environments are related to problems where multiple users and devices may interact, and where simple and local solutions could possibly generate good results, but may not be effective with regards to use and interaction.

There are many techniques that can be employed to face this kind of problems, from CORBA to multi-agent systems, passing by web-services and SOA, among others. All those methodologies have their advantages and disadvantages that are properly analyzed in this document, to finally explain the new architecture presented as a solution for distributed environment problems.

The new architecture for solving complex solutions in distributed environments presented here is called ***OBaMADE:*** *Organization Based Multiagent Architecture for Distributed Environments*. It is a multiagent architecture based on the organizations of agents paradigm, where the agents in the architecture are structured into organizations to improve their organizational capabilities.

The reasoning power of the architecture is based on the Case-Based Reasoning methodology, being implemented in an internal organization that uses agents to create services to solve the external requests made by the users.

The OBaMADE architecture has been successfully applied to two different case studies where its prediction capabilities have been properly checked. Those case studies have showed optimistic results and, being complex systems, have demonstrated the abstraction and generalizations capabilities of the architecture.

Nevertheless OBaMADE is intended to be able to solve much other kind of problems in distributed environments scenarios. It should be applied to other varieties of situations and to other knowledge fields to fully develop its potential.

Resvmen

Los entornos distribuidos representan un campo de conocimiento complejo en el que las soluciones a aplicar deben ser flexibles y deben contar con gran capacidad de adaptación. Este tipo de entornos está normalmente relacionado con problemas donde varios usuarios y dispositivos entran en juego. Para solucionar dichos problemas, pueden utilizarse sistemas locales que, aunque ofrezcan buenos resultados en términos de calidad de los mismos, no son tan efectivos en cuanto a la interacción y posibilidades de uso.

Existen múltiples técnicas que pueden ser empleadas para resolver este tipo de problemas, desde CORBA a sistemas multiagente, pasando por servicios web y SOA, entre otros. Todas estas metodologías tienen sus ventajas e inconvenientes, que se analizan en este documento, para explicar, finalmente, la nueva arquitectura presentada como una solución para los problemas generados en entornos distribuidos.

La nueva arquitectura presentada aquí se llama ***OBaMADE***, que es el acrónimo del inglés *Organization Based Multiagent Architecture for Distributed Environments (Arquitectura Multiagente Basada en Organizaciones para Entornos Distribuidos).* Se trata de una arquitectura

multiagente basada en el paradigma de las organizaciones de agente, donde los agentes que forman parte de la arquitectura se estructuran en organizaciones para mejorar sus capacidades organizativas.

La capacidad de razonamiento de la arquitectura está basada en la metodología de razonamiento basado en casos, que se ha implementado en una de las organizaciones internas de la arquitectura por medio de agentes que crean servicios que responden a las solicitudes externas de los usuarios.

La arquitectura OBaMADE se ha aplicado de forma exitosa a dos casos de estudio diferentes, en los que se han demostrado sus capacidades predictivas. Aplicando OBaMADE a estos casos de estudio se han obtenido resultados esperanzadores y, al ser sistemas complejos, se han demostrado las capacidades tanto de abstracción como de generalización de la arquitectura presentada.

Sin embargo, esta arquitectura está diseñada para poder ser aplicada a más tipo de problemas de entornos distribuidos. Debe ser aplicada a más variadas situaciones y a otros campos de conocimiento para desarrollar completamente el potencial de esta arquitectura.

Table of Contents

List of Figures

List of Tables

List of Algorithms

"Wisdom begins in wonder." *Socrates*

1. INTRODUCTION

This chapter briefly introduces the concepts treated in the remainder of the document. The main problems solved and the ways they are faced are explained first, allowing for a concise description of the elements that make result in the final solution proposed in this thesis. The methodology carried out all through the development of this document is also explained here. Finally, the structure of the whole document is also presented.

Distributed environments represent complex situations where multiple parameters are involved and where a series of different elements may interact. Those elements can be from the different persons implicated in the environment (that will be the users in a computer system) to the diverse external elements that must be taken into account when facing situations like those represented by distributed environments.

Artificial intelligence (AI) [Turing, 1950] have solved distributed problems applying its abilities and capabilities in different ways [Moulin and Chaib-Draa, 1996]. Various kinds of distributed systems operate today, each aimed at solving different kinds of problems. The challenges faced in building

a distributed system vary depending on the requirements of the system. In general, however, most systems will need to handle the following issues [Coulouris *et al.*, 2005, Van Steen and Tanenbaum, 2002].

- Various entities in the system must be able to *interoperate with one another*, despite differences in hardware architectures, operating systems, communication protocols, programming languages, software interfaces, security models, and data formats.
- *The entire system should appear as a single unit* and the complexity and interactions between the components should be typically hidden from the end user.
- *Failure* of one or more components should not bring down the entire system, and should be isolated.
- *Scalability*. The system should work efficiently with increasing number of users and addition of a resource should enhance the performance of the system.
- *Concurrency.* Shared access to resources should be made possible.
- *Openness and Extensibility*. Interfaces should be cleanly separated and publicly available to enable easy extensions to existing components and add new components.
- It is also important to allow the *movement of tasks within a system* without affecting the operation of users or applications, and distribute load among available resources for improving performance.
- *Security*. Access to resources should be secured to ensure only known users are able to perform allowed operations.

In this PhD Thesis document a new architecture to solve problems related with distributed environments is presented. It is called ***OBaMADE****: Organization Based Multiagent Architecture for Distributed Environments.* It is a multiagent architecture that is based on the organizations of agents

paradigm and that employs the Case-Based Reasoning (CBR) methodology [Watson and Marir, 1994] as the solution generation core.

1.1. HYPOTHESIS OF WORK AND MAIN OBJECTIVES

The fundamental hypothesis of this study is *to develop* ***an architecture to solve problems related with distributed environments.*** The architecture should face those problems offering *different interfaces to different users with different devices in a transparent way*. The architecture has to be based in *organizations of agents*. The agents that make those organizations must be designed as *dynamic agents*. The agents being part of the inner organizations, which are in charge of the generation of the solutions, should incorporate *reasoning mechanisms based on the Case-Based Reasoning methodology*. That methodology is based in the reuse of past information, adapting past solutions given to solve past problems to solve new problems arriving to the architecture. The solutions given to past problems are stored in the system related with the problems solved by those solutions.

To achieve the main hypothesis of this work it is necessary to analyze the state of the art of the distributed environments and its possible solutions, as well as agents and multi-agent systems (MAS) and organizations of agents. The main specific objectives that underlie the development of this architecture are:

- Make a study of the existing methodologies and technologies used to solve problems related with the distributed environments.
- Study the different approaches related with agents, multi-agent systems and organizations of agents and their evolutions, to properly choose the most appropriate one to be applied to this specific architecture.

- Apply the organization of agents theory to solve the distributed environment problem proposing an architecture that could be applied to solve different kind of problems in that kind of environments.
- Theoretically compare the advantages and disadvantages of the proposed architecture with the existing techniques and methodologies.
- Apply the presented architecture to real-life case studies, adapting the architecture to the problems by developing a prototype that could generate application results.
- Empirically evaluate the results obtained after applying the prototypes created based on the architecture to real-life environments, and comparing the results obtained with other existing techniques.

It is important to point out that the architecture generated in the investigation presented in this PhD. thesis is not only intended to solve the kinds of problems presented in the results section (natural distributed environments). The presented architecture is aimed at being able to adapt itself to different kinds of problems whose common characteristic is the existence of an underlying distributed environment.

1.2. METHODOLOGY

The investigation process followed in this PhD thesis uses the ActionResearch methodology. In this methodology the problem is first identified and then a hypothesis is proposed so that any further development will be based on that hypothesis. After the proposal, a compilation, organization and analysis of information is carried out, continuing with the design of a proposal focused to solve the problem. Finally, the conclusions are generated, after evaluating the results of the investigation. Six different activities were defined to follow this investigation model. They are necessary in order to achieve the objectives proposed.

First, *the problem to be solved and its main characteristics should be*

defined. This activity consists of the presentation of the problem, defining its characteristics and proposing a hypothesis to solve the problem totally or partially. The main objectives needed to solve the problem are also identified here. In this occasion the main objective is to design and construct an architecture to face distributed environment situations. The creation of an architecture of that type implies the analysis of the typical situations that will face. That analysis has implied the understanding of the inner characteristics of the distributed environments, which has helped to design the architecture presented here.

There should be an *actualization and complete revision of the state of the art*. The main areas, technologies and developments related with the present investigation are analyzed and the mayor developments in each of them are compiled. The state of the art is constantly revised, increasing the amount of information stored and considered. A theoretical layout is obtained that may enhance the knowledge and improve the development process. Focusing on distributed environments, in this investigation it has been necessary to analyze the different methodologies and technologies currently used to solve problems occurred in distributed environments. Once the organizations of agents theory was chosen as the one to be applied in the final design of the architecture, all the theory and applications of the agents, multi-agents systems and organizations of agents were analyzed.

The proposal should be gradually and iteratively designed and developed. Taking into account the information obtained in the previous activities, a model is designed and developed. That model integrates the components needed to generate a useful and innovative solution to the proposed problem. The solutions should achieve the objectives previously indicated. The architecture presented in this document has evolved from a simple local application, which could solve distributed problems in a quite restricted way, to a complex architecture formed by different organizations of

agents that collaborate to achieve a common aim, working together and exchanging information.

Incremental prototype systems should be created to experiment and implement the proposed solution. The functionalities, components, behaviours and interactions are formalized. Prototypes are developed to be implemented in specific application scenarios, within the scope of the problem, experimenting with those prototypes to obtain result data that will help to evaluate the proposed solution. The OBaMADE architecture has been applied to two different case studies. First, the oil spill problem where the architecture has been adapted to generate predictions of the situation of a specific oceanic area after an oil spill. Once the architecture demonstrated its validity applied to that problem, a second case study was chosen, applying the architecture to the case of the forest fires evolution prediction. In this occasion the architecture should forecast the situation of a forest area once a fire was nearby started.

The results achieved with the proposed solution must be analyzed and conclusions regarding those results must be formulated. A thorough analysis is done of the results obtained, evaluating the evolution of the outcomes through the development of the investigation. Conclusions are formulated, based on the initial hypothesis and the objectives achieved. The presented architecture has generated optimistic results after being applied to the two case studies cited before. In both situations, using historical data, the architecture has been able of generating precise and accurate predictions of the evolution of the oil slicks produced after an oil spill and of the fires in a forest environment.

The knowledge achieved, and also the results and experiences obtained should be constantly disseminated. This activity consists on the publication of contributions in journals, presentation of papers in conferences and workshops, revealing the advances and partial results of the investigation,

as well as the experience acquired through the development process. From the first steps of the investigation, where the designed architecture could face problems in distributed environments being a local software, it has been published both in journals [Mata and Corchado, 2009, Baruque *et al.*, 2010, Corchado *et al.*, 2010] and in different workshops and conferences[Corchado and Mata, 2008, Mata *et al.*, 2009], evolving to the final state presented in this document.

1.3. THESIS STRUCTURE

This document begins with the *introduction*, where the main elements covered by this thesis are briefly initiated. In this introduction, the main objectives and the central hypothesis of this work are briefly described. After introducing the main elements of the investigation, it is necessary to develop them, which is done following the structure explained next.

The architecture presented here is designed to work in distributed environments, so the first analysis done in this document was about the existing technologies applied to that kind of environments. This analysis is done in the *second chapter*, where the description of the distributed environments is presented, detailing the main features and the issues usually handled to face the problems originated in such environments. Then, the most important technologies applied to the distributed systems are explained. These include the following: CORBA, SOA, web services, grid computing and MAS.

To face the distributed environments, the architecture designed in this thesis uses organizations of agents, which are an evolution of the multi-agent systems focused on the organizational capabilities of those systems. So, prior to present the characteristics of the organizations of agents, the multi-agent systems where introduced in the *third chapter*. The explanation begins with the description of the concept of an agent and its attributes, followed by the

main characteristics of the multiagent systems and the agent societies, and ending with the coordination and communication of the multiagent systems.

As cited before, organizations of agents have been chosen to structure the architecture created through this PhD investigation. After explaining agents and multi-agent systems in the third chapter, the organizations of agents are exposed in the *fourth chapter*. Organizations of agents are a specific type of multi-agent system, where the agents forming part of the system follow a particular structure. The organizations of agents are based on human organizations, which are also explained at the beginning of this chapter. The main characteristics of the organizations are then described. Finally, the main types of organizations and their complete characteristics are specified in one of the *appendixes*: hierarchies, holarchies, coalitions, teams, congregations, societies, federations, markets, matrices and compound organizations.

Once introduced the main technologies used to solve distributed environments situations, and the ones used to design the architecture presented in this document, it is time to explain it, OBaMADE, which is done in the *fifth chapter* of this document. First, the main structure, composed of an interface organization, a communication organization and two service organizations is described. Then, the implemented reasoning services are detailed. Those services follow a CBR methodology in order to solve the problems to be faced with this architecture.

After explaining the main elements of the OBaMADE architecture it is necessary to check it, what is done in the *sixth chapter*. The OBaMADE architecture is applied to two different case studies. The first one is the oil spill problem, where there are different sources of information and different kinds of users that may interact with the system. The second case study is the application of the architecture to forest fires evolution prediction. This second use of the architecture is also dynamic and distributed with the involvement of different people. The forest fires problem serves as standard against which the

correction of the architecture is measured.

Finally, the model presented in this document is theoretically evaluated and both the final conclusions and future work are explained in the *seventh chapter*, presenting the conclusions and final analysis of the architecture as well as the intention for future work to be done based on the architecture developed.

Following the evolution of this document, a complete set of references walk alongside the different explanations done through the document. Those references are compiled after the conclusions and future work, in the *references* section. An important effort was required to compile such a vast selection of references (almost five hundred of them) related to the different parts of this document.

Finally, the appendixes have been included. They cover some technical explanations that could not be included in the main document. The *first appendix* is dedicated to explain the main elements and features of CORBA, one of the distributed environments techniques used for comparison with the OBaMADE architecture. The *appendix B* deeply explains the taxonomy of organizations which are the inspiration for the structure of the OBaMADE architecture. The *third appendix* is in charge of a complete explanation of the CBR methodology, which is used by OBaMADE to implements its internal solution generation services. The *final appendix* is a complete resume of the document in Spanish language.

"Nothing is particularly hard if you divide it into small jobs." *Henry Ford*

2. Distributed Environments

The architecture presented in this thesis mainly covers situations generated in distributed dynamic environments. In this chapter, the main characteristics of those systems are explained, as long as the existing solutions to face distributed environments, in the different possible situations that cover those kinds of environments. Finally, the main characteristics chosen to design the architecture presented in this thesis are exposed.

Several definitions and different points of view exist on what distributed systems are. Coulouris defines a distributed system as *"a system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing"* [Coulouris *et al.*, 2005]; and Tanenbaum defines it as *"A collection of independent computers that appear to the users of the system as a single computer"* [Van Steen and Tanenbaum, 2002]. Leslie Lamport – a famous researcher on timing, message ordering, and clock synchronization in distributed systems – once said that *"a distributed system is*

one on which I cannot get any work done because some machine I have never heard of has crashed" reflecting on the huge number of challenges faced by distributed system designers. Despite these challenges, the benefits of distributed systems and applications are many, making it worthwhile to pursue.

Various types of distributed systems and applications have been developed and are being used extensively in the real world. Here, the main characteristics of distributed systems are presented and look at some of the challenges that are faced by designers and implementers of such systems, and also introduce an example af a distributed system.

A common misconception among people when discussing distributed systems is that it is just another name for a network of computers. However, this overlooks an important distinction. A distributed system is built on top of a network and tries to hide the existence of multiple autonomous computers. It appears as a single entity providing the user with whatever services are required. A network is a medium for interconnecting entities (such as computers and devices) enabling the exchange of messages based on well-known protocols between these entities, which are explicitly addressable (using an IP address, for example).

In this chapter, first the distributed environment problem will be described defining the main characteristics of those environments. Then, after introducing the kind of problems to be solved, different existing approaches to solve them will be explained, including some of the techniques and methodologies most commonly used. The solutions to the distributed environment problems explained in this chapter are: CORBA, SOA, web services, grid computing and multiagent systems. Then, a brief introduction to the technologies employed in this investigation to design the proposed architecture is done.

2.1. PROBLEM DEFINITION

There are various types of distributed systems, such as Clusters [Buyya, 2002], Grids [Foster and Kesselman, 1999], P2P (Peer-to-Peer) networks [Subramanian and Goodman, 2005], distributed storage systems and so on. A cluster is a dedicated group of interconnected computers that appears as a single super-computer, generally used in high performance scientific engineering and business applications. A grid is a type of distributed system that enables coordinated sharing and aggregation of distributed, autonomous, heterogeneous resources based on users' QoS (Quality of Service) requirements. Grids are commonly used to support applications emerging in the areas of e-Science and e-Business, which commonly involve geographically distributed communities of people who engage in collaborative activities to solve large scale problems and require sharing of various resources such as computers, data, applications and scientific instruments. P2P networks are decentralized distributed systems, which enable applications such as file-sharing, instant messaging, online multi-user gaming and content distribution over public networks. Distributed storage systems such as NFS (Network File System) provide users with a unified view of data stored on different file systems and computers which may be on the same or different networks.

2.1.1. DISTRIBUTED SYSTEMS' MAIN FEATURES

There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of a distributed computing system is to connect users and resources in a transparent, open and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems.

The main features of a distributed system include [Coulouris *et al.*, 2005, Van Steen and Tanenbaum, 2002]:

- *Functional Separation:* based on the functionality/services provided, capability and purpose of each entity in the system.
- *Inherent distribution:* entities such as information, people, and systems are inherently distributed. For example, different information is created and maintained by different people. This information could be generated, stored, analysed and used by different systems or applications which may or may not be aware of the existence of the other entities in the system.
- *Reliability:* long term data preservation and backup (replication) at different locations.
- *Scalability:* addition of more resources to increase performance or availability.
- *Economy:* sharing of resources by many entities to help reduce the cost of ownership.

As a consequence of these features, the various entities in a distributed system can operate concurrently and possibly autonomously. Tasks are carried out independently and actions are co-ordinated at well-defined stages by exchanging messages. Also, entities are heterogeneous, and failures are independent. Generally, there is no single process, or entity, that has the knowledge of the entire state of the system.

2.1.2. MAIN ISSUES HANDLED BY DISTRIBUTED SYSTEMS

Various kinds of distributed systems operate today, each aimed at solving different kinds of problems. The challenges faced in building a distributed system vary depending on the requirements of the system. In

general, however, most systems will need to handle the following issues [Coulouris *et al.*, 2005, Van Steen and Tanenbaum, 2002]:

- *Heterogeneity:* various entities in the system must be able to interoperate with one another, despite differences in hardware architectures, operating systems, communication protocols, programming languages, software interfaces, security models, and data formats.
- *Transparency:* the entire system should appear as a single unit and the complexity and interactions between the components should be typically hidden from the end user.
- *Fault tolerance and failure management:* Failure of one or more components should not bring down the entire system, and should be isolated.
- *Scalability:* the system should work efficiently with increasing number of users and addition of a resource should enhance the performance of the system.
- *Concurrency:* shared access to resources should be made possible at the same time by different elements.
- *Openness and Extensibility:* interfaces should be clearly separated and publicly available to enable easy extensions to existing components and add new components by evolving the systems to a more complete state.
- *Migration and load balancing:* allow the movement of tasks within a system without affecting the operation of users or applications, and distribute load among available resources for improving performance.
- *Security:* access to resources should be secured to ensure only known users are able to perform allowed operations.

Several software companies and research institutions have developed distributed computing technologies that support some or all of the features described above.

2.1.3. DISTRIBUTED COMPUTING TECHNOLOGIES

Over the years, technologies such as CORBA and DCOM have provided the means to build distributed component-based systems. Such technologies allow systems to interoperate at the component level, by providing a software layer and protocols that offer the interoperability needed for components developed in different programming languages to exchange messages. However, such technologies present scalability issues when applied to, for instance, the Internet and some restrict the developer to a specific programming language. Hence, approaches based on Web protocols and XML (eXtensible Markup Language) have been proposed to allow interoperable distributed systems irrespective the programming language in which they are developed.

Web Services are based on XML and provide a means to develop distributed systems that follow a Service Oriented Architecture (SOA). Services are described in an XML-based dialect (WSDL). In a similar fashion, the request and reply messages exchanged in such systems are formatted according to the Simple Object Access Protocol (SOAP). SOAP messages can be encoded and transmitted by using Web protocols such as the Hypertext Transfer Protocol (HTTP). Various industrial technologies and application platforms such as .NET from Microsoft, J2EE from Sun, and WebSphere from IBM are targeted at supporting the development of applications based on Web Services.

Along with Web Services, Grid computing is another emerging paradigm for creating wide-area distributed applications. Web Services are foundation technologies that can be used in building many types of

distributed systems and applications including Grid systems. Web Services are in the core of the current implementations of Grid technologies such as Globus from Argonne National Laboratory in USA and the Gridbus from the University of Melbourne, Australia. Grid computing scales from an enterprise/organisation to a global level. Global Grids are established over the public Internet infrastructure, and are characterized by a global presence, comprise of highly heterogeneous resources, present sophisticated security mechanisms, focus on single sign-on and are mostly batch-job oriented.

To enable global Grids, one requirement is that current enterprise and campus Grids are able to interoperate. Enterprise and campus Grids consist of resources spread across an enterprise and provide services to users within that organisation and are managed by a single administrative domain. Such Grids are more concerned with cycle stealing from unused desktops and use virtualization of resources in order to provide better means to manage and utilize them within an enterprise. For example, Oracle 10g uses a virtualization approach to split data storage from the database transaction and process layer. However, scalability and the design of security mechanisms are not as difficult as they are for global Grids.

Next, some of those main technologies used to face different situations in distributed environments will be explained in detail.

2.2. CORBA

An important characteristic of large computer networks such as the Internet, the World Wide Web (WWW), and corporate intranets is that they are heterogeneous. For example, a corporate intranet might be made up of mainframes, UNIX workstations and servers, PC systems running various

flavours of Microsoft Windows, IBM OS/2, or Apple Macintosh, and perhaps even devices such as telephone switches, robotic arms, or manufacturing test beds. The networks and protocols underlying and connecting these systems might be just as diverse: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, and various remote procedure call (RPC) [Birrell and Nelson, 1984] systems, for example. Fundamentally, the rapidly-increasing extents of these networks are due to the need to share information and resources within and across diverse computing enterprises.

Heterogeneity in such computing systems is the result of several factors. The first one is engineering trade-offs. There is rarely only a single acceptable solution to a complex engineering problem. As a result, different people across an enterprise often choose different solutions to similar problems.

Cost effectiveness is also crucial. Vendors vary in their abilities to provide the *"best"* systems at the lowest cost. Though there is some amount of *"brand name loyalty"*, many consumers tend to buy the systems that best fulfil their requirements at the most reasonable price, regardless of who makes them.

Finally, legacy systems must be taken into account. Over time, purchasing decisions accumulate, and already-purchased systems may be too critical or too costly to replace. For example, a company that has been successfully running its order fulfilment applications, which are critical to its day-to-day operations, on its mainframe for the last fifteen years is not likely to simply scrap their system and replace it with the latest fad technologies. Alternatively, a company may have spent large sums of money on its current systems, and those systems must be utilized until the investment has paid off.

Ideally, heterogeneity and open systems enable to use the best combination of hardware and software components for each portion of an enterprise. When the right standards for interoperability and portability

between these components are in place, the integration of the components yields a system that is coherent and operational.

Unfortunately, dealing with heterogeneity in distributed computing enterprises is rarely easy. In particular, the development of software applications and components that support and make efficient use of heterogeneous networked systems is very challenging. Many programming interfaces and packages currently exist to help ease the burden of developing software for a single homogeneous platform. However, few help deal with the integration of separately-developed systems in a distributed heterogeneous environment.

In recognition of these problems, the Object Management Group (OMG) was formed in 1989 to develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environments. Since that time, the OMG has grown to become the largest software consortium in the world, with over 700 developers, vendors, and end users on its membership roster. These members contribute technology and ideas in response to Requests For Proposals (RFPs) issued by the OMG. Through responses to these RFPs, the OMG adopts specifications based on commercially-available object technology. Here the OMG's *Object Management Architecture* (OMA) [OMG, 1996] is described, focusing on one of its key components, the *Common Object Request Broker Architecture* (CORBA) specification [OMG, 1996].

In this chapter, only the main elements of CORBA are going to be described, analyzing the interest of this methodology to solve the problems generated in distributed environments. The rest of the technical explanation of CORBA will be developed in *Appendix A*, where a complete description will be held.

2.2.1. THE OBJECT MANAGEMENT ARCHITECTURE (OMA)

The OMA [OMG, 1996] is composed of an Object Model and a Reference Model. The Object Model defines how objects distributed across a heterogeneous environment can be described, while the Reference Model characterizes interactions between those objects. The OMG RFP process is used to adopt technology specifications that fit into the Object Model and the Reference Model and work with the other previously-adopted specifications. Through adherence to the OMA, these specifications allow for the development and deployment of interoperable distributed object systems in heterogeneous environments.

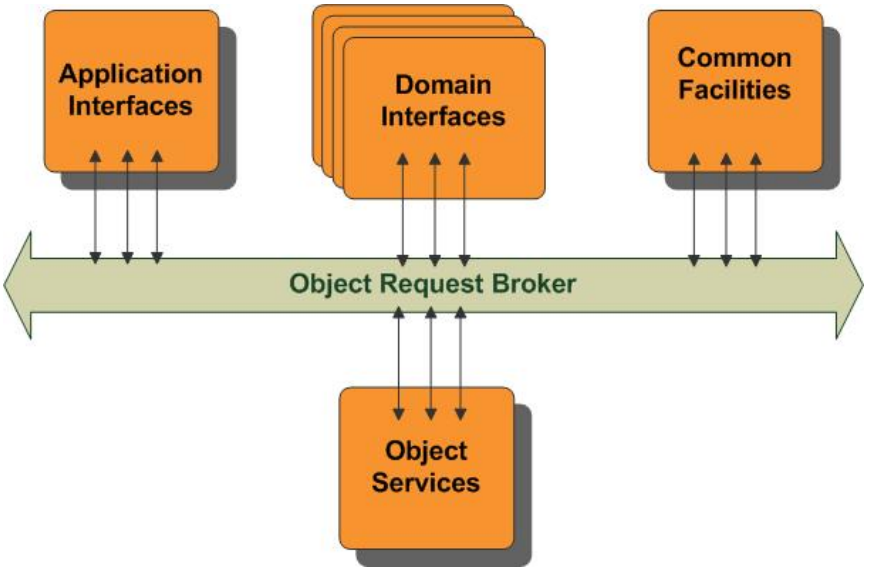


Figure 1. *OMA Reference Model Interface Categories.*

In the OMA Object Model, an object is an encapsulated entity with a distinct immutable identity whose services can be accessed only through well-defined interfaces. Clients issue requests to objects to perform

services on their behalf. The implementation and location of each object are hidden from the requesting client.

Figure 1 shows the components of the OMA Reference Model. The *Object Request Broker* (ORB) component is mainly responsible for facilitating communication between clients and objects. Utilizing the ORB component are four object interface categories:

- *Object Services (OS):* these are domain-independent interfaces that are used by many distributed object programs. For example, a service providing for the discovery of other available services is almost always necessary regardless of the application domain. Two examples of Object Services that fulfil this role are:
  - *The Naming Service* – which allows clients to find objects based on names.
  - *The Trading Service* – which allows clients to find objects based on their properties.

  There are also Object Service specifications for lifecycle management, security, transactions, and event notification, as well as many others [OMG, 1995b].
- *Common Facilities (CF):* like Object Service interfaces, these interfaces are also horizontally-oriented, but unlike Object Services they are oriented towards end-user applications. An example of such a facility is the Distributed Document Component Facility (DDCF) [OMG, 1995a], a compound document Common Facility based on OpenDoc. DDCF allows for the presentation and interchange of objects based on a document model, for example, facilitating the linking of a spreadsheet object into a report document.

– *Domain Interfaces (DF):* these interfaces fill roles similar to Object Services and Common Facilities but are oriented towards specific application domains. For example, one of the first OMG RFPs issued for Domain Interfaces is for Product Data Management (PDM) Enablers for the manufacturing domain. Other OMG RFPs will soon be or already have been issued in the telecommunications, medical, and financial domains. In *figure 2*, multiple boxes are shown for Domain Interfaces to indicate the existence of many separate application domains.

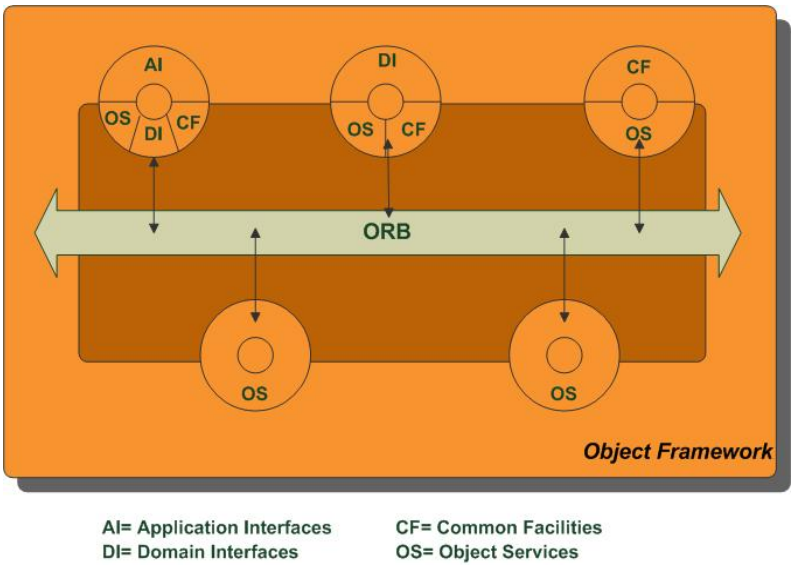


Figure 2. *OMA Reference Model Interface Usage.*

– *Application Interfaces (AI):* these are interfaces developed specifically for a given application. Because they are application-specific, and because the OMG does not develop applications (only specifications), these interfaces are not standardized. However, if over time it appears that certain broadly useful services emerge out of a particular application domain, they might become candidates

for future OMG standardization.

Figure 2 illustrates the other part of the OMA Reference Model, the concept of *Object Frameworks*. These are domain-specific groups of objects that interact to provide a customizable solution within that application domain. These frameworks are typically oriented towards domains such as telecommunications [Siegel, 1998], medical systems [Moreno *et al.*, 2008], finance [Jian-dong and Shang-liang, 2007], and manufacturing [Lai, 2007]. In *figure 2,* each circle represents a component that uses the ORB to communicate with other components.

The interfaces supported by each component are indicated on its outer circle. As *figure 2* shows, some components support application-specific interfaces, as well as domain interfaces, common facilities interfaces, and object services. Other components support only a subset of these interfaces.

Within an object framework like the one shown in *figure 2*, each component communicates with others on a *peer-to-peer* basis. That is, each component is both a *client* of other services and a *server* for the services it provides. In CORBA, the terms "client" and "server" are merely roles that are filled on a per-request basis. Very often, a client for one request is the server for another.

Throughout most of its existence, much of the OMG's attention was focused on the ORB component of the OMA. This was necessary because everything else in the OMA depends on the ORB.

2.2.2. CORBA APPLICATIONS AND INTEREST FIELDS

Areas that are currently being investigated by OMG task forces include:

- *Medical (Master Patient Indexing):* patient identification can be surprisingly difficult, due to multiple people with the same name,

illegal use of identification numbers, etc. At the time of this document has been written, the CORBAmed Medical Task Force was very close to issuing a RFP for technology related to the identification of patients.

- *Telecommunications (Isochronous Streams):* streams for audio and video data have special quality of service requirements due to their isochronous nature. The CORBAtel Telecommunications Task Force recently issued an RFP seeking technology for the management and manipulation of isochronous streams.
- *Business (Business Objects):* portions of many business processes are very similar, and thus can be abstracted out into frameworks. The Business Objects Task Force will soon begin evaluating responses to its Business Objects RFP, which seeks object frameworks to support business processes.
- *Common Facilities (Systems Management Facility):* the OMG has nearly completed the adoption of the X/Open systems management specification, which defines a set of extended services for the monitoring and management of distributed systems. These services complement those specified in the existing OMG Common Object Services Lifecycle Specification [OMG, 1995b].
- *ORBOS (Objects by value):* CORBA currently allows object references to be passed as arguments and return values, but it does not allow objects to be passed by value. This makes the use of encapsulated data types (e.g., linked lists) difficult to use from languages such as C++. The ORBOS Task Force will soon begin evaluating responses to its Objects by Value RFP, which will describe technology for passing objects by value between CORBA applications.

2.3. SOA

Over the last four decades, software architectures have attempted to deal with increasing levels of software complexity. As the level of complexity continues to evolve, traditional architectures do not seem to be capable of dealing with the current problems. While traditional needs of IT organizations persist, the need to both respond quickly to new requirements of the business and continually reduce the IT cost, and the ability to absorb and integrate new business partners and new customer sets become more in demand. The industry has gone through multiple computing architectures designed to allow fully distributed processing, programming languages designed to run on any platform, greatly reducing implementation schedules, and a myriad of connectivity products designed to allow better and faster integration of applications. Service Oriented Architecture (SOA) is being advocated in the industry as the next evolutionary step in software architecture to help IT organizations meet their ever more complex set of challenges [Channabasavaiah *et al.*, 2003].

The existence of Web services technologies has stimulated the discussion of Services Oriented Architecture (SOA), which has been advocated for more than a decade now, ever since CORBA extended the promise of integrating applications on disparate heterogeneous platforms. Problems of integrating those applications arise, often because of so many different (and non-CORBA-compliant) object models. Architects and engineers alike became so bogged down in solving technology problems, constantly in search for a more robust architecture that would allow simple, fast, and secure/seamless integration of systems and applications was lost. Meanwhile, the distributing computing model opens the way of cross-platform and cross-programming language interoperability. SOAP is a great distribution computing solution because it achieves interoperability through open

standards at the specification level as well as the implementation level.

Meanwhile, basic business needs such as lowering costs, reducing cycle times, integration across the enterprise, B2B and B2C integration, greater ROI, creating an adaptive and responsive business model demands better solutions. *"Point solutions"* won't work as desired solutions for the lack of a consistent architectural framework within which applications can be rapidly developed, integrated, and reused. Thus an architectural framework must be developed to allow the assembly of components and services for the rapid, and even dynamic, delivery of solutions; an architectural view unconstrained by technology.

2.3.1. DEFINITION OF SOA

A service-oriented architecture is essentially *a collection of services, among which the communication can involve either simple data passing or it could involve two or more services coordinating some activity, requiring means of connecting services to each other* [Krafzig *et al.*, 2004]. The first service-oriented architecture in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification.

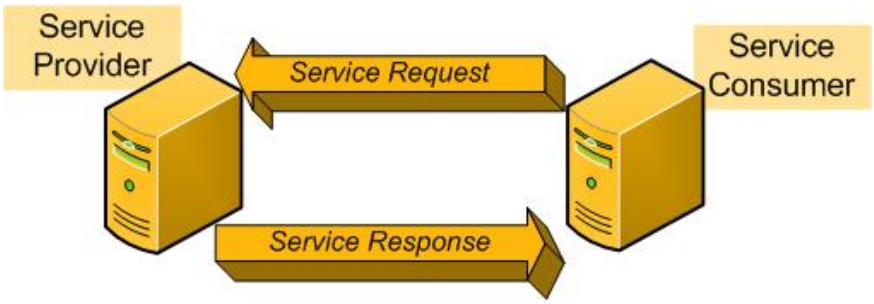


Figure 3. *Basic Service-Oriented Architecture.*

To understand service-oriented architecture must begin with a clear understanding of the term service. A service is a function that is well defined, self-contained, and does not depend on the context or state of other services. The technology of Web services is the most likely connection technology of service-oriented architectures. Web services essentially use XML to create a robust connection.

Figure 3 illustrates a basic service-oriented architecture. It shows a *service consumer* at the right sending a *service request* message to a *service provider* at the left. The *service provider* returns a *response message* to the *service consumer*. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. How those connections are defined is explained in Web Services explanation [Erickson and Siau, 2008]. A service provider can also be a service consumer.

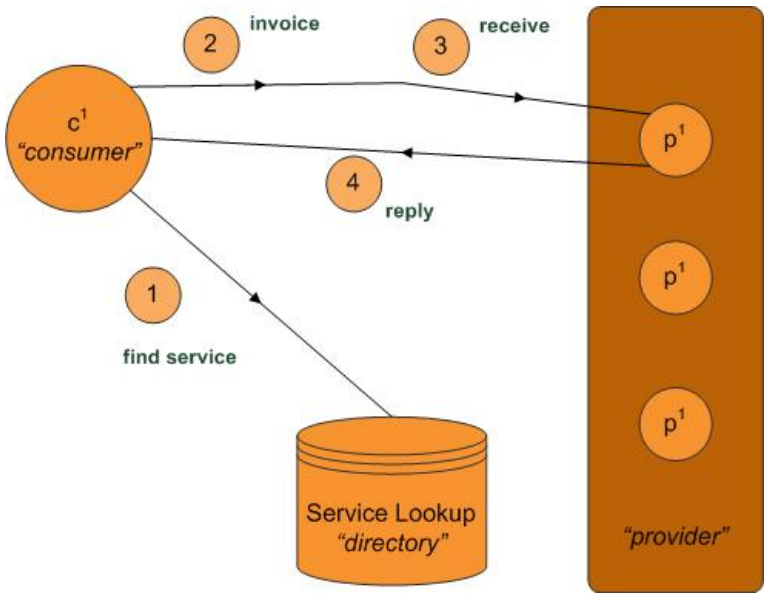


***Figure 4**.Components of basic Service-Oriented Architecture.*

As a distributed software model, a SOA is usually comprised of three primary parts: *provider* (of services), *consumer* (of services) and *directory* (of services), as shown in *figure 4*. Web Services are considered an example of Service Oriented Architecture. Service Networks take on the properties of a SOA.

Considering the term *service-oriented architecture*, it is useful to review the key terms, as it is done in the following paragraphs.

An architecture is a formal description of a system, defining its purpose, functions, externally visible properties and interfaces. It also includes the description of the system's internal components and their relationships, along with the principles governing its design, operation, and evolution.

A service is a software component that can be accessed via a network to provide functionality to a service requester.

The term *service-oriented architecture* refers to a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.

Technically, then, the term SOA refers to the *design* of a system, not to its implementation. It is common place for the term to be used in referring to an implementation. For example, in phrases such as *"building a SOA"* and using the adjective *service-oriented* in contexts such as *"service-oriented environment"* or *"service-oriented grid"*.

SOA is considered as an *architectural style* that emphasizes implementation of components as modular *services* that can be discovered and used by clients [Mahmoud, 2005].

Services *may be individually useful*, or they can be integrated (composed) to provide higher-level services. Among other benefits, this promotes reuse of existing functionality. Services *communicate with their*

clients by exchanging messages. They are defined by the messages they can accept and the responses they can give. Services *can participate in a workflow*, where the order in which messages are sent and received affects the outcome of the operations performed by a service. This notion is defined as "service choreography".

Services *may be completely self-contained*, or they *may depend on the availability of other services, or on the existence of a resource such as a database*. In the simplest case, a service might perform a calculation such as computing the cube root of a supplied number without needing to refer to any external resource, or it may have pre-loaded all the data that it needs for its lifetime.

Conversely, a service that performs currency conversion would need real-time access to exchange-rate information in order to yield correct values. Services *advertise details such as their capabilities, interfaces, policies, and supported communications protocols*. Implementation details such as programming language and hosting platform are of no concern to clients, and are not revealed.

Figure 5 illustrates a simple service interaction cycle, which begins with a *service advertising* itself through a well-known *registry service* (1). A potential *client*, who may or may not be another service, queries the *registry* (2) to search for a service that meets its needs. The registry returns a (possibly empty) list of suitable services, and the client selects one and passes a request message to it, using any mutually recognized *protocol* (3). In this example, the service *responds* (4) either with the result of the requested operation or with a fault message.

The illustration shows the simplest case, but in a real-world setting such as a commercial application the process may be significantly more complex. For example, a given service may support only the HTTPS protocol, be restricted to authorized users, require Kerberos authentication,

offer different levels of performance to different users, or require payment for use.

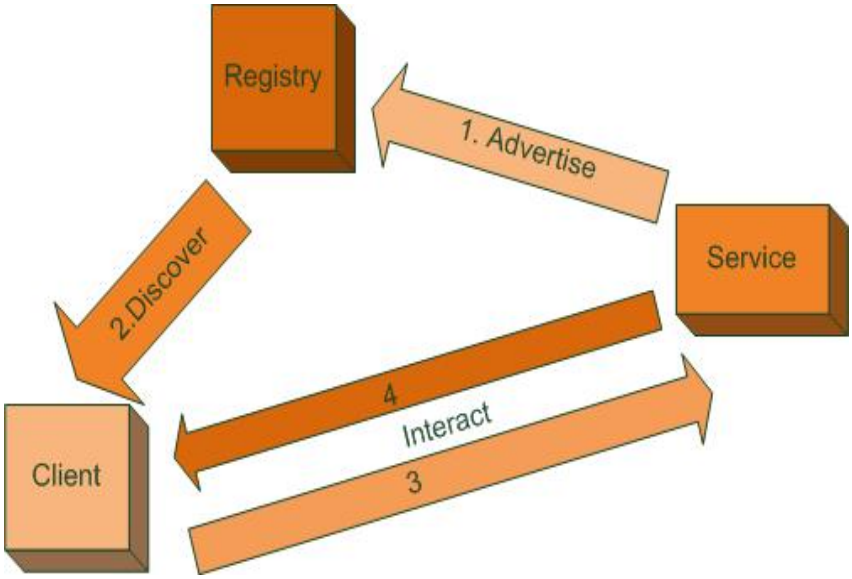


Figure 5. *Service interaction in a service-oriented environment.*

Services can provide such details in a variety of ways, and the client can use this information to make its selection. Some attributes, such as payment terms and guaranteed levels of service, may need to be established by a process of negotiation before the client can make use of the service it has selected.

And, while this illustration shows a simple synchronous, bi-directional message exchange pattern, a variety of patterns are possible. For example, an interaction may be one-way, or the response may come not from the service to which the client sent the request, but from some other service that completed the transaction.

2.3.2. LOOSE COUPLING

When talking about and defining SOA, the term *loose coupling* is included [Natis and Schulte, 2003]. This term implies that the interacting software components minimize their in-built knowledge of each other: they discover the information they need at the time they need it. For example, having learned about a service's existence, a client can discover its capabilities, its policies, its location, its interfaces and its supported protocols. Once it has this knowledge, the client can access the service using any mutually acceptable protocol. The word "frictionless" has been used to describe the ultimate goal of loose coupling, and the word aptly conjures up a vision of components that communicate almost without contact. The benefits of loose coupling include:

- *Flexibility:* a service can be located on any server, and relocated as necessary. As long as it maintains its registry entry, prospective clients will be able to find it.
- *Scalability:* services can be added and removed as demand varies.
- *Replaceability:* Provided that the original interfaces are preserved, a new or updated implementation of a service can be introduced, and outdated implementations can be retired, without disruption to users.
- *Fault tolerance:* if a server, a software component, or a network segment fails, or the service becomes unavailable for any other reason, clients can query the registry for alternate services that offer the required functionality, and continue to operate without interruption.

Clearly, all these benefits have great value in a dynamic distributed environment. However, while the vision of loose coupling is appealing, it is some way from broad-based realization. For example, common Web

service integrated development environments (IDEs) provide for rapid and easy development of service clients by reading the description of a service and generating a client-side *"proxy"* or *"stub"* class with methods that correspond to the service's interfaces. If the interfaces change, the proxy must be regenerated and the client code may need to be altered to invoke the changed methods. While development in this type of environment may be fast and easy, the result is far from frictionless.

Does this mean that services and clients built using such an IDE are not loosely coupled? Well, the word *"loose"* is presumably chosen because it is a relative term. It might be said that a truly frictionless relationship is zero-coupled, and adding some friction simply moves it further toward the other end of the scale. The point at which it becomes tightly coupled is a subjective decision.

2.3.3. STATE AND STATELESSNESS

A key notion of loose coupling is *statelessness*, which is a topic that has been much-discussed and is often mentioned as a critical requirement, sometimes without a clear understanding of its significance [Stal *et al.*, 2006].

Simply, the benefits of loose coupling, as listed above, are derived from the fact that a client can choose to go to any service that is capable of fulfilling its need. If its choice is restricted to a single service then a tight coupling exists between the client and the server, and the benefits of loose coupling are diminished.

In the simple case of a calculator or a stock-price service it is easy to see that once a client has requested and received information, the transaction is completed, and the client has no particular need to revisit the same service for its future needs. From this perspective, the client and service are loosely coupled.

For a more complex transaction that requires several steps, however, the design of the service might be such that the service retains in its local memory some information (*"state"*) about the first step, expecting to make use of it when the client contacts it for the next step. In this case, the service is *"stateful"*, and the client must return to the same service for the next step. This might result in a delay if many clients are using the same service or in a transaction failure if the node hosting the service fails between steps.

A better approach to the design of the service not to retain the state about the transaction, to be *"stateless"*. This implies that in a multi-step transaction, at the end of each intermediate step, the service must hand back to the client sufficient state information to enable any qualified service to identify and continue the transaction. The client must hand the state information to whichever service it selects to process the next step of the transaction.

The selected service must be able to accept and handle the state information supplied by the client, regardless of whether it processed the earlier steps itself.

Figure 6 shows a client engaged in a three-step transaction with several services, each of which might be capable of handling any part or all of the transaction. The service that handles Step 1 stores the details of the in-progress transaction in the database, and returns requested information to the client, along with a transaction identifier. The client might request confirmation from the user before passing the transaction identifier to another service, which uses it to retrieve the state information from the database and initiates Step 2. This service then updates the database and returns additional information to the client. Finally, the client passes the transaction identifier back to a third service with a request to complete the transaction.

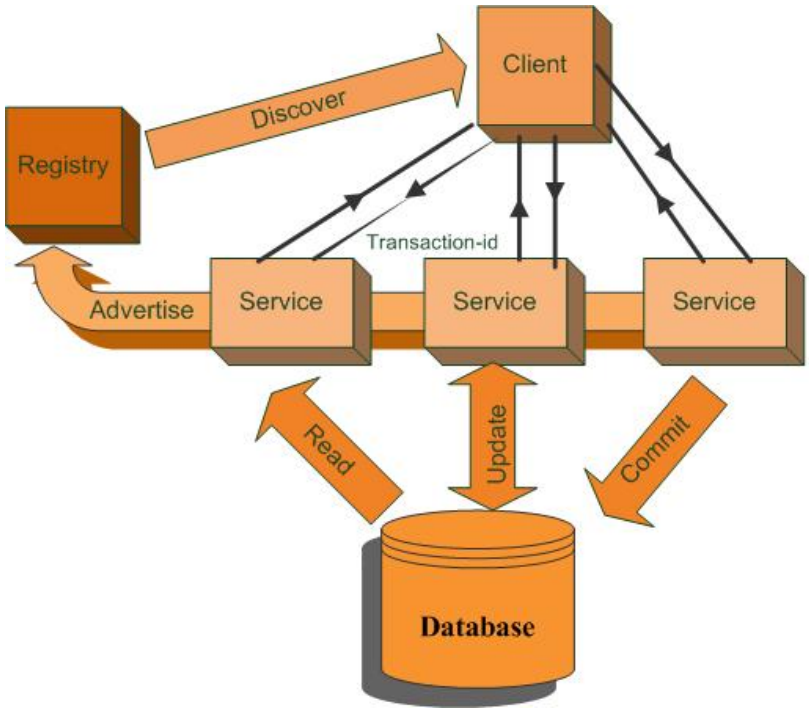


Figure 6. *A multi-step client/service interaction.*

Most non-trivial applications require access to some amount of state information, and the debate is not so much about *whether* state should exist as about *where* it should be stored. The approach outlined above enhances loose coupling by separating the transaction's state from the services that operate on it. In the example, both the account data and the details of the transaction can be considered to be state information, but the account data is permanent, while the transaction details only need to exist while the transaction is in progress. To minimize the amount of state that needs to be passed between the clients and the services, the critical account data and the details of the transaction are held in the database: the common requirement for all participating services is that they must be able to access the database, given a simple token such as a customer's account number, which can easily be passed between the client and the services.

2.3.4. SERVICE-ORIENTED ARCHITECTURE (SOA) MODEL

The potential concept of SOA was found to have merit by companies like IBM and Microsoft who recognized that for SOA to succeed where other distributed computing concepts had failed, it must be implemented on open standards. Thus, the recent cooperation between these companies on recommended standards like UDDI and WSDL [Schroth and Christ, 2007]. According to IBM, SOA is comprised of three participants and three fundamental operations, regardless of its implementation, (see *Figure 7*).

A *service provider* is a network node that provides a service interface for a software asset that manages a specific set of tasks. A service provider node can represent the services of a business entity or it can simply represent the service interface for a reusable subsystem.

A *service requestor* is a network node that discovers and invokes other software services to provide a business solution. Service requestor nodes will often represent a business application component that performs remote procedure calls to a distributed object, the service provider. In some cases, the provider node may reside locally within an intranet or in other cases it could reside remotely over the Internet. The conceptual nature of SOA leaves the networking, transport protocol, and security details to the specific implementation.

The *service broker* is a network node that acts as a repository, yellow pages, or clearing house for software interfaces that are published by service providers. A business entity or an independent operator can represent a service broker.

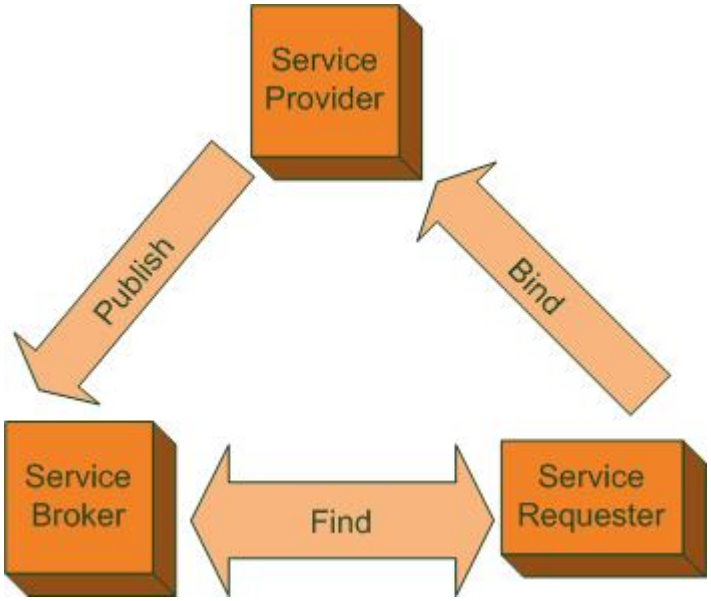


Figure 7. *The SOA model.*

These three SOA participants interact through three basic operations: *publish*, *find*, and *bind*. Service providers *publish* services to a service broker. Service requesters *find* required services using a service broker and *bind* to them. The interactive process among these three agents calls/centres on the *service* components (rather than objects which characterizes object paradigm).

2.3.5. BUSINESS ROLES

Because of the role-based nature, SOA strives to meet services and business needs much more effectively. In the service-oriented architecture (SOA) of Web services, three distinct actors the *provider*, the *requestor*, and the *broker* interact to help an organization make a choice among five possible business roles [Pasley, 2005].

- *Service Requestor:* for a business to identify with this SOA role, it must find some commonality between their business activity and

the actions of a requestor. There are two clear business activities that would allow a business to benefit from implementing the role of a service requestor: Content Aggregation and Service Aggregation. Content Aggregation is an activity where a business entity interacts with a variety of content providers to process or reproduce such content in the desired presentation format of its customers (such as Internet portal or information service provider). Service Aggregation is an activity where a business entity interacts with a variety of service providers to re-brand, host, or offer a composite of services to its customers (such as a mobile portal and the alike of OnStar).

- *Service Provider:* for a business to identify with this SOA role, it must view itself as performing some degree of an electronic service. Whether that service is defined as the processing of data or the act of carrying out a specific task, the business entity must believe it is performing work for others as an occupation or a business.
- *Registry:* if a business entity finds itself collecting and cataloguing data about other businesses and then selling that data to others, it may identify well with a registry, a form of SOA Broker. Usually, a registry would collect data such as business name, description, and contact information. In UDDI terms, this SOA role is often referred to as the White Pages.
- *Broker:* building on the concept of a registry, business entities may also be able to identify with the notion of a broker, which in UDDI terms is often referred to as Yellow Pages. Brokers usually extend the value proposition of a registry by offering intelligent search capability and business classification or taxonomy data.

- *Aggregator/Gateway:* any business entity that provides Broker capabilities plus the ability to describe actual policy, business processes and binding descriptions would be able to identify itself as *Green Pages.*

2.4. WEB SERVICES

In recent years, distributed programming paradigms have emerged, that allow generic software components to be developed and shared. Whilst early versions were little more than shared libraries of functions with little user documentation and unpredictable side effects, it wasn't until the advent of object-oriented programming and architectures such as CORBA, that self contained components could be reliably defined, documented and shared within a distributed environment. Although ideal for some enterprise integration and eCommerce, it has only been with the adoption of XML as common data syntax that the underlying principals have gained wide scale adoption, through the definition of Web Service standards. *Web services are well defined, reusable, software components that perform specific, encapsulated tasks via standardized Web-oriented mechanisms*. They can be discovered, invoked, and the composition of several services can be choreographed, using well defined workflow modelling frameworks.

Whilst promising to revolutionize eCommerce and enterprise-wide integration, current standard technologies for Web services (e.g. WSDL [Christensen *et al.*, 2001]) provide only syntactic-level descriptions of their functionalities, without any formal definition to what the syntactic definitions might mean. In many cases, Web services offer little more than a formally defined invocation interface, with some human oriented metadata that describes what the service does, and which organization developed it (e.g. through UDDI descriptions). Applications may invoke Web services using a common, extendable communication framework (e.g. SOAP). However, the

lack of machine readable semantics necessitates human intervention for automated service discovery and composition within open systems, thus hampering their usage in complex business contexts.

Semantic Web Services (SWS) relax this restriction by augmenting Web services with rich formal descriptions of their capabilities, thus facilitating automated composition, discovery, dynamic binding and invocation of services within an open environment. A prerequisite to this, however, is the emergence and evolution of the Semantic Web, which provides the infrastructure for the semantic interoperability of Web Services. Web Services will be augmented with rich formal descriptions of their capabilities, such that they can be utilized by applications or other services without human assistance or highly constrained agreements on interfaces or protocols. Thus, Semantic Web Services have the potential to change the way knowledge and business services are consumed and provided on the Web.

Current efforts in developing Semantic Web Service infrastructures can be characterized along three orthogonal dimensions: usage activities, architecture and service ontology. Usage activities define the functional requirements, which a framework for Semantic Web Services ought to support. The architecture of SWS describes the components needed for accomplishing the activities defined for SWS, whereas the service ontology aggregates all concept models related to the description of a Semantic Web Service.

2.4.1. DEFINITION

Web Services are changing the way applications communicate with each other on the Web. They promise to integrate business operations, reduce the time and cost of Web application development and maintenance as well as promote reuse of code over the World Wide Web. By allowing functionality to be encapsulated and defined in a reusable

standardized format, Web services have enabled businesses to share (or trade) functionality with arbitrary numbers of partners, without having to prenegotiate communication mechanisms or syntax representations. The advent of discovery has enabled vendors to search for Web services, which can then be invoked as necessary. For example, a book-selling company may look for shipping services, which they may later invoke to ensure that books are delivered to the customers. This flexibility is achieved through a set of well-defined standards that define syntax, communication protocol, and invocation signatures, which allow programs implemented on diverse, heterogeneous platforms to interoperate over the internet.

A Web Service is a software program identified by an URI (Uniform Resource Identifier), which can be accessed via the internet through its exposed interface. The interface description declares the operations which can be performed by the service, the types of messages being exchanged during the interaction with the service, and the physical location of ports, where information should be exchanged. For example, a Web service for calculating the exchange rate between two money currencies can declare the operation *"getEx changeRate"* with two inputs of type string (for source and target currencies) and an output of type float (for the resulting rate). A binding then defines the machine and ports where messages should be sent. Although there can be many ways of implementing Web services, it is basically assumed that they are deployed in Web servers such that they can be invoked by any Web application or Web agent independently of their implementations. In addition Web services can invoke other Web services.

The common usage scenario for Web services can be defined by three phases: *Publish*, *Find* and *Bind*, and three entities: the *service requester*, which invokes services, the *service provider* which responds to

requests, and the *registry* where services can be published or advertised. A service provider publishes a description of a service it provides to a service registry. This description (or advertisement) includes a profile on the provider of the service (e.g. company name and address); a profile about the service itself (e.g. name, category), and the URL of its service interface definition (e.g. WSDL description).

When a developer realizes a need for a new service, he finds the desired service either by constructing a query, or browsing the registry. The developer then interprets the meaning of the interface description (typically through the use of meaningful label or variable names, comments, or additional documentation) and binds to (i.e. includes a call to invoke) the discovered service within the application they are developing. This application is known as the service requester. At this point, the service requester can automatically invoke the discovered service (provided by the service provider) using Web service communication protocols (e.g. SOAP).

Key to the interoperation of Web services is an adoption of a set of enabling standard protocols. Several XML-based standards have been proposed to support the usage scenario previously described.

XML schema (XML-S) [Biron and Malhotra, 2001] provides the underlying framework for both defining the Web Services Standards, and variables, objects and data types etc that are exchanged between services. SOAP [Mitra, 2003] is W3C's recommended XML-data transport protocol, used for data exchange over Web-based communications protocols (http). SOAP messages can carry an XML payload defined using XML-S, thus ensuring a consistent interpretation of data items between different services.

WSDL [Christensen *et al.*, 2001] is the W3C recommended language for describing the service interface. Two levels of abstraction are

used to describe Web services. The first level defines atomic method calls, or *operations*, in terms of input and output *messages* (each of which contain one or more parameters defined in XML-S). Operations define the way in which messages are handled e.g. whether an operation is a *one-way operation*, *request-response*, *solicit-response* or *notification*. The second abstraction maps operations and associated messages to physical endpoints, in terms of *ports* and *bindings*. Ports declare the operations available with corresponding inputs and outputs. The bindings declare the transport mechanism (usually SOAP) being used by each operation. WSDL also specifies one or more network locations or endpoints at which the service can be invoked.

As services become available, they may be registered with a UDDI registry [Dialani, 2002] which can subsequently be browsed and queried by other users, services and applications. UDDI Web service discovery is typically human oriented, based upon yellow or white-page queries (i.e. metadata descriptions of service types, or information about the service providers). UDDI service registrations may also include references to WSDL descriptions, which may facilitate limited automation of discovery and invocation. However, as no explicit semantic information is normally defined, automated comprehension of the WSDL description is limited to cases where the provider and requester assume pre-agreed ontologies, protocols and shared knowledge about operations.

A service might be defined as a workflow describing the choreography of several operations. Such a workflow may determine: the order of operation execution, what operations may be executed concurrently and alternative execution pathways (if conditional operators are included in the workflow modelling language). Conversely, workflows are required to orchestrate the execution of several simple services that may be composed together to form a more complex service. Various

choreography and orchestration languages have been proposed such as BPEL4WS [Andrews *et al.*, 2003], and are currently being evaluated by various industry standardization bodies.

2.4.2. WEB SERVICES PROBLEMS

SOAP, WSDL, and UDDI are important technologies to enable Web services. However, to fully satisfy the requirements of business applications, the current technologies have shortcomings. Here, the three major problems and research directions to upgrade the existing technologies will be discussed.

2.4.2.1. SECURITY PROBLEMS

Now, a simple travel scenario will be used to illustrate the security problem of Web services. More than three pieces of the Web services framework are required to interact properly to complete the travel scenario.

At the very least, it is necessary to ensure that transactions like the electronic check-ins were conducted in a secure environment and that messages were reliably delivered to the destinations. The main reason to built additional security when there are technologies such as Secure Multipurpose Internet Mail Extensions (S-MIME), HTTP Secure (HTTPS), and Kerberos available is the difference between end-to-end and single-hop usage.

Business messages typically originate from one application and then are transferred to another one. Mechanisms such as Secure Sockets Layer are great for securing (for confidentiality) a direct connection from one machine to another, but they are of no help if the message has to travel over more than one connection.

It is well known in the penetration testing community that attacks to modern systems are usually not at the network level but within the application protocols (e.g., HTTP in the case of Web systems). This means that the firewall will simply pass the attack traffic along with any legitimate HTTP requests as it looks for port 80 traffic only, and does not concern the malformed HTTP traffic or application specific attacks (such as SQL injection). In many cases where SSL is used, the firewall cannot see into the traffic stream. In some respects, Web services have adopted the HTTP's tunnelling idea, by allowing all systems, both internal and external, to communicate on HTTP ports so flexibility is obtained. What is removed is the control the firewall may have, and ultimately the application servers are opened up to "application level" attacks in exactly the same way as insecure and vulnerable Web servers today.

Basically, the security problems that are likely to affect Web services are the same as those that have affected the conventional Web-based systems. Security is critical to the adoption of Web services by enterprises, but, as it stands today, the Web services framework does not meet basic security requirements.

The fact that Web services involve exchange of messages means that securing the message exchange is an important issue to consider when building and using Web services. In the Web services context, security means that the recipient of a message should be able to verify the integrity of the message and to make sure that it has not been modified. The recipient should have received a message confidentially so that unauthorized users could not read it, know the identity of the sender and determine whether or not the centre is authorized to carry out the operation requested in the message. These are usually met through encrypting messages.

On the other hand, because Web services allow all systems, both internal and external, to communicate on HTTP ports, the application servers are inevitably opened up to "application level" attacks.

A few standards have come out to alleviate the message security problem, including WS Security and various other initiatives (mostly from the major vendors and PKI suppliers) towards enabling digital signatures on XML messages and transactions. But the "application level" attacks were hardly concerned.

2.4.2.2. COMPOSITION PROBLEMS

Complex business interactions require support for higher levels of business functionality. Business interactions are typically long execution processes and involve multiple interactions between partners. To deploy and effectively use these types of services, it is necessary to represent business processes and states of services and to create service compositions (complex aggregations) in a standardized and systematic fashion. Several proposals for accomplishing this task exist; see, for example, Web Services Flow Language, XLANG [Thatte, 2001] and BPEL4WS.

The industry has used a number of terms to describe how components can be connected together to build complex business processes. Workflow and document management systems have existed as a means to handle the routing of work between various resources in an IT organization. These resources might include people, systems or applications and typically involve some human intervention. Business process management systems (BPMS) have also been used to enable a business to build a top-down process design model, consisting of various integration activities (e.g., integration to a legacy system).

BPMS systems [Lee *et al.*, 2002] would typically cover the full lifecycle of a business process, including modelling, executing, monitoring, management and optimization tasks. With the introduction of Web services, terms such as *"Web services composition"* and *"Web services flow"* were used to describe the composition of Web services in a process flow. More recently, the terms *orchestration* and *choreography* have been used to describe this too. Orchestration describes how Web services can interact with each other at the message level, including business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional, multi-step process model.

2.4.2.3. SEMANTIC PROBLEMS

The current Web services technology basically provides a syntactical solution and still lacks the semantic part. A Web service is described in WSDL, outlining what input the service expects and what output it returns. To exploit their potentials (beyond the enterprise application integration), Web services must be able to orchestrate themselves into more complex services. Thus, methods to combine individual Web services into a distributed, higher-level service are needed. The Web Service Flow Language (WSFL), which can express the sequencing of individual services, is taking the first steps. WSFL lets the user decide which Web services to combine and in what order. However, a framework that semantically describes services so that software agents can locate, identify and combine these services is still needed.

Many researchers believe that the Semantic Web vision of the next-generation Web, that enables computers unambiguously

interpreting the Web content, addresses precisely this problem [Gibbins *et al.*, 2004, Hendler, 2001, McIlraith and Zeng, 2001]. The Semantic Web project is Tim Berners-Lee's brainchild, seeking to create a machine processable Web. Semantic Web has advocates predominantly from the more research-oriented members of the Web community. Due to commercial interests, industrial player, including Microsoft, IBM and BEA, on the other hand, have largely driven the development of Web Services.

In his opening lecture at the Twelfth International World Wide Web conference, the Director of the World Wide Web Consortium explained how to make the two main thrusts of the development of the Web not compete, but work together. Berners-Lee claimed that Web Services meet immediate technology needs, while the Semantic Web has the potential for future exponential growth. There are many ways in which the two areas could interact in the future, and the W3C does not intend to limit their work to one area or the other.

Current Web services standards, such as SOAP, WSDL, XLANG, WSFL, BPEL4WS, WSCI and BPML describe Web service content in terms of XML syntax. Unfortunately, XML alone lacks both a well-defined semantics and sufficient expressive power to realize the vision of diverse Web services having wide-scale interoperability. Seamless interoperability between services that have not been pre-designed to work together requires programs to describe their own capabilities and understand other services' capabilities. To realize this vision, Web content, particularly Web service content and capabilities, may need to be described in a language that goes beyond XML. This problem is well addressed in the Semantic Web vision of the next-generation Web.

2.5. GRID COMPUTING

The main goal in describing the Grid architecture is not to provide a complete enumeration of all required protocols (and services, APIs, and SDKs) but rather to identify requirements for general classes of component. The result is an extensible, open architectural structure within which can be placed solutions to key VO (*Virtual Organization*) requirements. The architecture described here and the subsequent discussion organizes components into layers, as shown in *figure 8.* Components within each layer share common characteristics but can build on capabilities and behaviours provided by any lower layer.

In specifying the various layers of the Grid architecture, the principles of the "hourglass model" [Kleinrock, 1994] are followed. The neck of the hourglass defines a fundamental set of core abstractions and protocols, onto which many different high-level behaviours can be mapped (the top of the hourglass), and which themselves can be mapped onto many different underlying technologies (the base of the hourglass).

By definition, the number of protocols defined at the neck must be small. In this architecture, the neck of the hourglass consists of *Resource* and *Connectivity* protocols, which facilitate the sharing of individual resources. Protocols at these layers are designed so that they can be implemented on top of a diverse range of resource types, defined at the *Fabric* layer, and can in turn be used to construct a wide range of global services and application-specific behaviours at the *Collective* layer. *Figure 8* shows that, because the Internet protocol architecture extends from network to application, there is a mapping from *Grid* layers into *Internet* layers. The architectural description is high level and places few constraints on design and implementation.

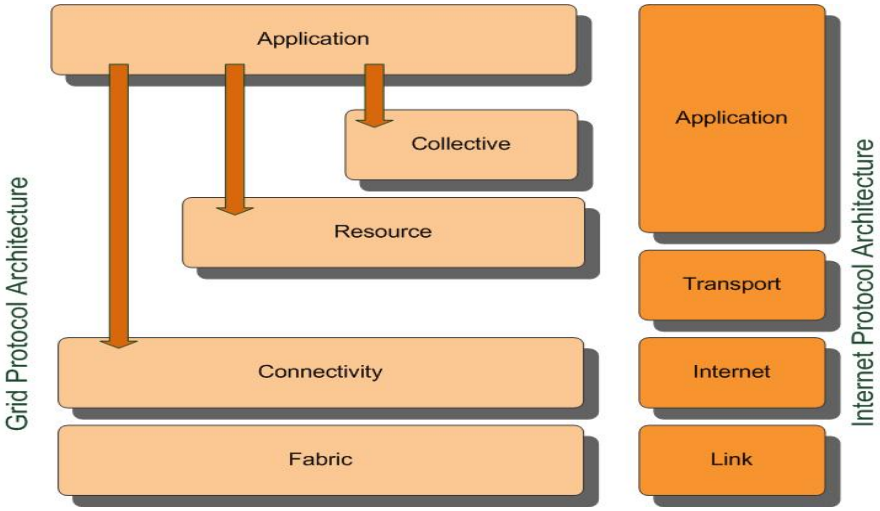


Figure 8. *The layered Grid architecture and its relationship to the Internet protocol architecture.*

2.5.1. INTERFACES TO LOCAL CONTROL

The *Grid Fabric* layer provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogues, network resources, and sensors. A "resource" may be a logical entity, such as a distributed file system, computer cluster, or distributed computer pool; in such cases, a resource implementation may involve internal protocols (e.g., the NFS storage access protocol or a cluster resource management system's process management protocol), but these are not the concern of Grid architecture.

Fabric components implement the local, resource-specific operations that occur on specific resources (whether physical or logical) as a result of sharing operations at higher levels. There is thus a tight and subtle interdependence between the functions implemented at the Fabric level, on the one hand, and the sharing operations supported, on the other. Richer *Fabric* functionality enables more sophisticated sharing operations;

at the same time, placing few demands on Fabric elements, then deployment of Grid infrastructure is simplified. For example, if resources support advance reservations, then it is straightforward to implement higher-level services that co-schedule multiple resources. However, as in practice few resources support advance reservation *"out of the box"*, a requirement for advance reservation increases the cost of incorporating new resources into a Grid.

Experience suggests that at a minimum, resources should implement enquiry mechanisms that permit discovery of their structure and state, on the one hand, and resource management mechanisms that provide some control of delivered quality of service, on the other. The following brief and partial list provides a resource-specific characterization of capabilities.

- *Computational resources:* mechanisms are required for starting programs and for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as current load and queue state in the case of scheduler-managed resources.
- *Storage resources:* mechanisms are also required for putting and getting files. Third-party and high-performance (e.g., striped) transfers are useful [Thompson *et al.*, 1999]. So are mechanisms for reading and writing subsets of a file and/or executing remote data selection or reduction functions [Beynon *et al.*, 2000]. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are useful, as are advance reservation

mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

- *Network resources:* management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.
- *Code repositories:* this specialized form of storage resource requires mechanisms for managing versioned source and object code: for example, a control system such as CVS.
- *Catalogues:* this specialized form of storage resource requires mechanisms for implementing catalogue query and update operations: for example, a relational database [Baru *et al.*, 1998].

2.5.2. CONNECTIVITY: COMMUNICATING EASILY AND SECURELY

The *Connectivity* layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources.

Communication requirements include transport, routing and naming. While alternatives certainly exist, in almost all practical situations these protocols will be drawn from the TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet layered protocol architecture [Baker, 1995].

With respect to security aspects of the *Connectivity* layer, it can be observed that the complexity of the security problem makes it important that any solutions be based on existing standards whenever possible. As with communication, many of the security standards developed within the context of the Internet protocol suite are applicable.

Authentication solutions for Virtual Organizations (VO) environments should have the following characteristics [Butler *et al.*, 2000]:

- *Single sign on:* users must be able to "log on" (authenticate) just once and then have access to multiple Grid resources defined in the Fabric layer, without further user intervention.
- *Delegation* [Foster *et al.*, 1998, Gamma *et al.*, 1995, Howell *et al.*, 2000]: a user must be able to endow a program with the ability to run on that user's behalf, so that the program is able to access the resources on which the user is authorized. The program should (optionally) also be able to conditionally delegate a subset of its rights to another program (sometimes referred to as restricted delegation).
- *Integration with various local security solutions:* each site or resource provider may employ any of a variety of local security solutions, including Kerberos and UNIX security. Grid security solutions must be able to interoperate with these various local solutions. They cannot, realistically, require wholesale replacement of local security solutions but rather must allow mapping into the local environment.
- *User-based trust relationships:* in order for a user to use resources from multiple providers together, the security system must not require each of the resource providers to cooperate or interact with

each other in configuring the security environment. For example, if a user has the right to use sites A and B, the user should be able to use sites A and B together without requiring that A's and B's security administrators interact.

Grid security solutions should also provide flexible support for communication protection (e.g., control over the degree of protection, independent data unit protection for unreliable protocols, and support for reliable transport protocols other than TCP) and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.

2.5.3. RESOURCE: SHARING SINGLE RESOURCES

The *Resource* layer builds on *Connectivity* layer communication and authentication protocols to define protocols (and APIs and SDKs) for the secure initiation, monitoring, and control of sharing operations on individual resources. *Resource* layer implementations of these protocols call *Fabric* layer functions to access and control local resources. *Resource* layer protocols are concerned entirely with individual resources and hence ignore issues of global state and atomic actions across distributed collections; such issues are the concern of the *Collective* layer discussed next.

Two primary classes of *Resource* layer protocols can be distinguished:

- *Information protocols* are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy.
- *Management protocols* are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to

be performed, such as process creation, or data access. Since management protocols are responsible for instantiating sharing relationships, they must serve as a *"policy application point"*, ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling (for example, terminating) the operation.

While many such protocols can be imagined, the Resource (and Connectivity) protocol layers form the neck of the hourglass model, and as such, it is required a small and standard set. These protocols must be chosen so as to capture the fundamental mechanisms of sharing across many different resource types (for example, different local resource management systems), while not overly constraining the types or performance of higher-level protocols that may be developed.

2.5.4. COLLECTIVE: COORDINATING MULTIPLE RESOURCES

While the Resource layer is focused on interactions with a single resource, the next layer in the architecture contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource but rather are global in nature and capture interactions across collections of resources. For this reason, the next layer of the architecture is named as the Collective layer. Because Collective components build on the narrow Resource and Connectivity layer *"neck"* in the protocol hourglass, they can implement a wide variety of sharing behaviours without placing new requirements on the resources being shared.

Directory services allow VO participants to discover the existence and/or properties of VO resources. A directory service may allow its users

to query for resources by name and/or by attributes such as type, availability, or load.

Co-allocation, scheduling, and brokering services allow VO participants to request the allocation of one or more resources for a specific purpose and the scheduling of tasks on the appropriate resources. Examples include AppLeS [Berman, 1999, Berman *et al.*, 1996], Condor-G, Nimrod-G [Abramson *et al.*, 1995], and the DRM broker [Beiriger *et al.*, 2000].

Monitoring and diagnostics services support the monitoring of VO resources for failure, adversarial attack (*"intrusion detection"*), overload, and so forth.

Data replication services support the management of VO storage (and perhaps also network and computing) resources to maximize data access performance with respect to metrics such as response time, reliability, and cost [Allcock *et al.*, 2001, Hoschek *et al.*, 2000].

Grid-enabled programming systems enable familiar programming models to be used in Grid environments, using various Grid services to address resource discovery, security, resource allocation, and other concerns. Examples include Grid-enabled implementations of the Message Passing Interface [Foster and Karonis, 1998, Gabriel *et al.*, 1998] and manager-worker frameworks [Casanova *et al.*, 2000, Goux *et al.*, 2000].

Software discovery services discover and select the best software implementation and execution platform based on the parameters of the problem being solved [Casanova et al., 1998]. Examples include NetSolve [Casanova and Dongarra, 1997] and Ninf [Nakada et al., 1999].

Community authorization servers enforce community policies governing resource access, generating capabilities that community members can use to access community resources. These servers provide a global policy enforcement service by building on resource information,

and resource management protocols (in the Resource layer) and security protocols in the Connectivity layer. Akenti [Thompson *et al.*, 1999] addresses some of these issues.

Collaboratory services support the coordinated exchange of information within potentially large user communities, whether synchronously or asynchronously. Examples are CAVERNsoft [DeFanti and Stevens, 1999, Leigh *et al.*, 1997], Access Grid [Childers *et al.*, 2000], and commodity groupware systems.

These examples illustrate the wide variety of *Collective* layer protocols and services that are encountered in practice. Notice that while *Resource* layer protocols must be general in nature and are widely deployed, *Collective* layer protocols span the spectrum from general purpose to highly application or domain specific, with the latter existing perhaps only within specific VOs.

Collective functions can be implemented as persistent services, with associated protocols, or as SDKs (with associated APIs) designed to be linked with applications. In both cases, their implementation can build on *Resource* layer (or other *Collective* layer) protocols and APIs. For example, *figure 9* shows a *Collective* co-allocation API and SDK (the middle tier) that uses a Resource layer management protocol to manipulate underlying resources. Above this, a co-reservation service protocol is defined and implements a co-reservation service that speaks this protocol, calling the co-allocation API to implement co-allocation operations and perhaps providing additional functionality, such as authorization, fault tolerance, and logging. An application might then use the co-allocation service protocol to request end-to-end network reservations.

Collective components may be tailored to the requirements of a specific user community, VO, or application domain, for example, an

SDK that implements an application-specific coherency protocol, or a co-reservation service for a specific set of network resources. Other *Collective* components can be more general-purpose, for example, a replication service that manages an international collection of storage systems for multiple communities, or a directory service designed to enable the discovery of VOs. In general, the larger the target user community, the more important it is that a *Collective* component's protocol(s) and API(s) be standards based.

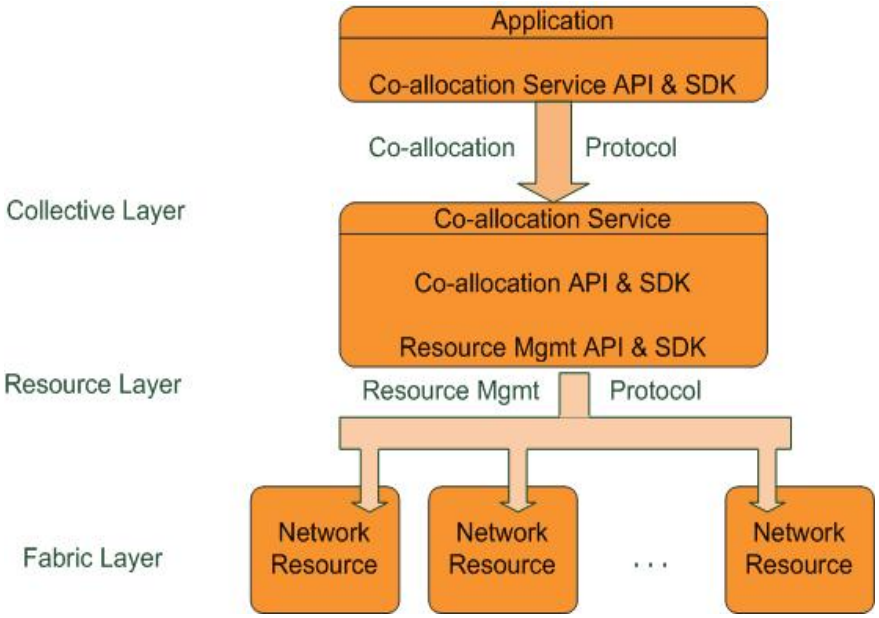


Figure 9. *Collective and Resource layer protocolos, service, APIs and SDKs.*

2.5.5. APPLICATIONS

The final layer in the Grid architecture comprises the user applications that operate within a VO environment. *Figure 9* illustrates an application programmer's view of Grid architecture. Applications are

constructed in terms of, and by calling upon, services defined at any layer. At each layer, there are well-defined protocols that provide access to some useful service: resource management, data access, resource discovery, and so forth. At each layer, APIs may also be defined whose implementation (ideally provided by third-party SDKs) exchange protocol messages with the appropriate service(s) to perform desired actions.

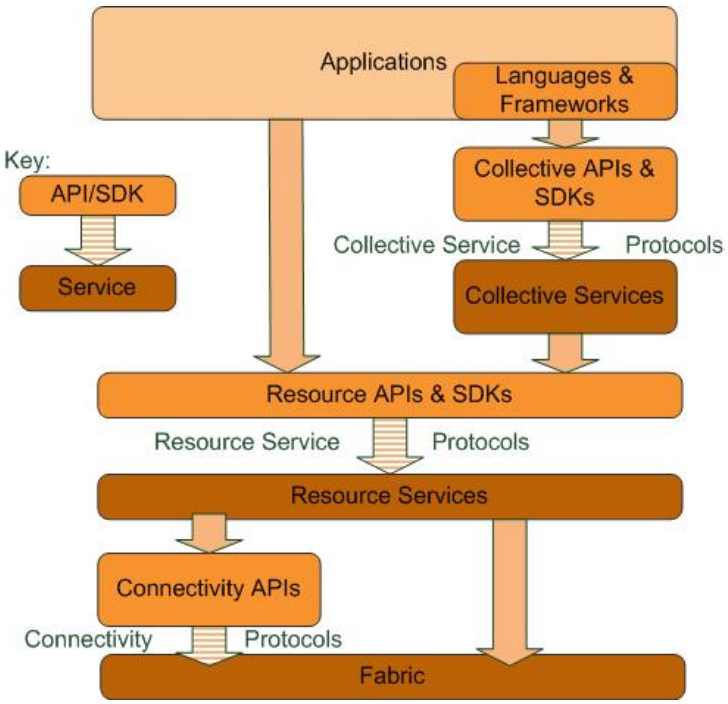


Figure 10. *Software development kits (SDKs) implement specific APIs.*

Figure 10 shows the implemented specific APIs use *Grid protocols* to interact with network services that provide capabilities to the end user. Higher level SDKs can provide functionality that is not directly mapped to a specific protocol, but may combine protocol operations with calls to additional APIs as well as implement local functionality. Notice the additional *"Languages and Frameworks"* component introduced in

figure 10. While the preceding discussion has focused on protocols as a means of achieving interoperability and APIs as a way of promoting code sharing and portability, effective application development can often benefit from the use of higher-level languages and frameworks (e.g., the Common Component Architecture [Armstrong *et al.*, 1999], SciRun [Casanova *et al.*, 1998], CORBA [Gannon and Grimshaw, 1998], [López *et al.*, 2000], Legion [Grinshaw and Wm, 1996], Cactus [Benger *et al.*, 1999]). These higher-level systems can build on protocols, services and APIs provided within the Grid architecture.

2.5.6. CURRENT DEVELOPMENTS AND LIMITATIONS

The infrastructure that focuses on management of distributed application data is commonly labelled a Data Grid [Chervenak *et al.*, 2000]. An increasing number of scientific disciplines manage large data collections generated by measurements and derivation of measurement data. As a result, many Data Grids are currently being deployed [Avery and Foster, 2000], [Avery *et al.*, 2001]. Infrastructure targeting resource information is often referred to as a Grid Information Service [Czajkowski *et al.*, 2001]. A number of research groups have designed and prototyped components for collecting, indexing and publishing Grid information. The problems of indexing, discovering, and accessing such *"Grid information services"* is in some respects quite similar to those encountered when indexing, discovery and accessing other data sources.

For both infrastructures, appropriate *data schemas* must be defined so that information can be encoded, stored and searched in an efficient manner. A number of recent developments have made contributions in that area. In the Data Grid context, the Chimera system [Foster *et al.*, 2002] targets a data schema that can be used to establish a

virtual data catalogue that describes all ways in which data in the catalogue has been derived. This is a generic solution that should be applicable to many different VOs and has been demonstrated for high-energy physics and astronomy applications. In the context of Grid Information Services, schemas are being developed for various Grid resource types as part of the GGF activities in the Grid Information Services working group. Commonalities with Common Information Model (CIM) are also being explored.

The definition of schemas is an important, but in some sense mundane, issue. More challenging is the design and implementation of a distributed system that implements mechanisms to publish information, disseminate information, notify participant of information changes, locate information, and retrieve information. Initial Grid infrastructure efforts have engineered software solutions for those mechanisms (e.g. [Fitzgerald *et al.*, 1997]). Those mechanisms have made it possible to take the first steps in Grid computing and have been crucial to making the Grid a plausible platform. However, a large part of those efforts were focused on *"getting it to work"*, without directly addressing issues of scalability, reliability and information quality.

Now, to face VOs that contain thousands of individuals in hundreds of institutions world-wide, issues such as scalability and usability are becoming a near-term concern. These issues are being increasingly recognized by the Grid computing community and recent work explores avenues of research that are strongly connected to distributed systems and distributed computing research questions. In that sense, Grid computing presents a key opportunity for distributed systems and distributed computing researchers. Grid developers are implementing large scale infrastructures such as GriPhyn, and those infrastructures provide a great *"playground"* to explore research issues in a concrete

setting that will have a major impact on disciplinary science. Furthermore, information dissemination techniques developed in the distributed systems community (e.g. wide-area group communications) have shortcomings that must be addressed for Grid computing.

2.6. AGENTS AND MULTIAGENT SYSTEMS

It is necessary to begin by defining an agent. What actually constitutes an agent, and how it differs from a normal program, has been heavily debated for several years now. While this debate is by no means over, there are a lot of agents loosely defined as programs that assist people and act on their behalf. This is what it is better to call the *"end-user perspective"* of software agents.

Considering an *end-user perspective*, an agent can be defined as *a program that assists people and acts on their behalf. Agents function by allowing people to delegate work to them.*

While this definition is basically correct, it does not really get under the hood. Agents come in myriad different types and in many settings. They can be found in computer operating systems, networks, databases, and so on. What properties do these agents share that constitute the essence of being an *agent*?

This is not the place to examine the characteristics of the numerous agent systems made available to the public by many research labs. But if you looked at all these systems, you would find that a property shared by all agents is that fact that they live in some environment. They have the ability to interact with their execution environment, and to act asynchronously and autonomously upon it. No one is required either to deliver information to the agent or to consume any of its output. The agent simply acts continuously in pursuit of its own goals.

In contrast to software objects of object-oriented programming, agents

are active entities that work according to the so-called *Hollywood Principle*: "*Don't call us, we'll call you!*"

Considering a *system's perspective*, an agent can be defines as: *a software object that*

- ✓ is situated within an execution environment;
- ✓ possesses the following mandatory properties:
  - *Reactive:* senses changes in the environment and acts accordingly to those changes;
  - *Autonomous:* has control over its own actions;
  - *Goal driven:* is pro-active;
  - *Temporally continuous:* is continuously executing;
- ✓ and may possess any of the following orthogonal properties:
  - *Communicative:* able to communicate with other agents;
  - *Mobile:* can travel from one host to another;
  - *Learning:* adapts in accordance with previous experience;
  - *Believable*: appears believable to the end-user.

2.6.1.MULTI-AGENT SYSTEMS

A multi-agent system (MAS) [Wooldridge, 2002] is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. Examples of problems which are appropriate to multi-agent systems research include online trading, disaster response, and modelling social structures.

MAS systems tend to find the best solution for their problems *"without intervention"*. There is high similarity here to physical phenomena, such as energy minimizing, where physical objects tend to

reach the lowest energy possible, within the physical constrained world. For example: many of the cars entering a metropolis in the morning will be available for leaving that same metropolis in the evening.

It would be foolish to claim that MAS should be used when designing all complex systems. Like any useful approach, there are some situations for which it is particularly appropriate, and others for which it is not.

The most important reason to use MAS when designing a system is that some domains require it. In particular, if there are different people or organizations with different (possibly conflicting) goals and proprietary information, then a multiagent system is needed to handle their interactions. Even if each organization wants to model its internal affairs with a single system, the organizations will not give authority to any single person to build a system that represents them all: the different organizations will need their own systems that reflect their capabilities and priorities.

For example, consider a manufacturing scenario in which company X produces tires, but subcontracts the production of lug-nuts to company Y. In order to build a single system to automate (certain aspects of) the production process, the internals of both companies X and Y must be modelled. However, neither company is likely to want to relinquish information and/or control to a system designer representing the other company. Perhaps with just two companies involved, an agreement could be reached, but with several companies involved, MAS is necessary. The only feasible solution is to allow the various companies to create their own agents that accurately represent their goals and interests. They must then be combined into a multiagent system with the aid of some of the techniques described here.

Another example of a domain that requires MAS is an hospital scheduling as presented in Decker's and Li's system [Decker and Li, 1998]. This domain from an actual case study requires different agents to represent the interests of different people within the hospital. Hospital employees have different interests, from nurses who want to minimize the patient's time in the hospital, to x-ray operators who want to maximize the throughput on their machines. Since different people evaluate candidate schedules with different criteria, they must be represented by separate agents if their interests are to be justly considered.

Even in domains that could conceivably use systems that are not distributed, there are several possible reasons to use MAS. Having multiple agents could speed up a system's operation by providing a method for parallel computation. For instance, a domain that is easily broken into components--several independent tasks that can be handled by separate agents--could benefit from MAS. Furthermore, the parallelism of MAS can help deal with limitations imposed by time-bounded reasoning requirements.

While parallelism is achieved by assigning different tasks or abilities to different agents, robustness is a benefit of MAS that have redundant agents. If control and responsibilities are sufficiently shared among different agents, the system can tolerate failures by one or more of the agents. Domains that must degrade gracefully are in particular need of this feature of MAS: if a single entity--processor or agent--controls everything, then the entire system could crash if there is a single failure. Although a MAS does not need to be implemented on multiple processors, to provide full robustness against failure, its agents should be distributed across several machines.

Another benefit of MAS is their scalability. Since they are inherently modular, it should be easier to add new agents to a multiagent

system than it is to add new capabilities to a monolithic system. Systems whose capabilities and parameters are likely to need to change over time or across agents can also benefit from this advantage of MAS.

From a programmer's perspective the modularity of multiagent systems can lead to simpler programming. Rather than tackling the whole task with a centralized agent, programmers can identify subtasks and assign control of those subtasks to different agents. The difficult problem of splitting a single agent's time among different parts of a task solves itself. Thus, when the choice is between using a multiagent system or a single-agent system, MAS is often the simpler option. Of course there are some domains that are more naturally approached from an omniscient perspective--because a global view is given--or with centralized control--because no parallel actions are possible and there is no action uncertainty. Single-agent systems should be used in such cases.

Agent and multi-agent systems will be deeply explained in next chapter, beginning with the main characteristics of a single agent, and passing to the organizational characteristics that share the agents within a multi-agent system.

2.7. SUMMARY AND CONCLUSIONS

After explaining the main methodologies used to face the problems generated by the distributed models, now, the structure of the model explained in this document are going to be detailed. The methodologies explained before, show that there are a great variety of different approaches to cope with the different circumstances which source is the intrinsic characteristics of the distributed environments.

As it was explained before, agents represent the most flexible way to solve problems originated by distributed environments. Specially, when

treating different sources of information, simultaneous request and when it is necessary to be adaptable to different kind of problems. In this occasion, an organization of agents has been chosen to create this new architecture. The main reason to choose an organization of agents as the structure of this architecture is that it is a very open way of organizing heterogeneous elements as those that make part of this architecture (interfaces, communication agents and services).

The organization of agents represents the internal structure of the presented architecture. On the other hand there are a series of services that implement the different services that cover the phases of a Case-Based Reasoning cycle, used to treat the information introduced in the system, and to generate the solutions to the different proposed problems. Those services are requested from the interface agents through the internal communication structure.

Both elements (the organization of agents and the CBR services) will be explained in the fifth chapter, where the OBaMADE architecture is fully explicated.

In this second chapter, different approximations to the distributed environments have been explained. First, the main characteristics of the distributed environments have been explained, taking special attention in the problematic aspects of those environments, and the difficulties that those aspects generate in order to face those situations.

After describing the issues handled by distributed systems, the different methodologies used to cope with that kind of systems have been explained. The methodologies chosen to be explained have been the following: C*ORBA, SOA, Web services, Grid computing and Multiagent systems.*

These techniques represent current approaches to solve the distributed environment problems. As explained in the previous subsection, the

architecture proposes here uses some of them introducing the organizations of agents and the case-based reasoning methodology as novelties. The combination of those methodologies with some artificial intelligence (AI) techniques [Gale, 2009, Haupt *et al.*, 2008] produces a powerful architecture that may be applied to different scenarios.

After having explained those technologies, the methodologies used in the architecture presented in this document will be explained in detail. First, multiagent systems will be specified, including the main characteristics of the agents themselves, and the composition of multiagent systems. Then, the organizations of agents will be described, starting with the concept of organization and finishing with a complete classification of organizations.

"There are no great limits to growth because there are no limits of human intelligence, imagination, and wonder." *Ronald Reagan*

3. Agents and MultiAgent Systems

Within this chapter, the general concept of agency will be elaborated upon. First of all, the question of what makes an agent to be an agent is discussed. Having identified the crucial requirements for agenthood, several different attributes associated with the scientific considerations of agents will be discussed in this chapter. This basic information will support the understanding of the particular features of interacting, intelligent agents.

The major issues confronting users of increasingly complex knowledge and information systems include access and availability of information and knowledge resources, confidence in the veracity and applicability of information provided, and assessment of the trustworthiness of the provider [Klusch, 1999]. Intelligent agents are a new paradigm for developing software applications and are

currently the focus of intense interest on the part of several fields of computer science and artificial intelligence [Jennings *et al.*, 1998]. Agents have made it possible to support the representation, coordination, and cooperation between heterogeneous processes and their users. A growing number of researchers and organizations are using agents in an increasingly wide variety of applications. Current *'real world'* agent applications cover several domains in industry, commerce, health care and entertainment, and range from comparatively small systems such as e-mail filters to large, open, complex, mission critical systems such as air traffic control.

Agents represent an intuitive way to solve distributed problems such the ones solved through the investigation reflected in this document. As explained in the previous chapter, distributed environments generate quite complex problems that must be solved with appropriated methodologies and technologies. Agents are one of the approaches commonly used to solve distributed environment problems. Agents are the basic element that structures the architecture presented in this document. As it will be explained in next chapters, agents can arrange themselves into organizations that help to achieve the objectives they were designed to accomplish. But first it is necessary to introduce the main concepts regarding agents, their attributes and how they can interact with each other. That is what will be explained in this chapter, paying special attention to the benefits of the agents to face distributed situations. The associative capabilities of the agents are also considered in this chapter, as an advance to the organizations of agents, which will be deeply explained in next chapter and in an appendix.

3.1. AGENTS THEORY

As already introduced in the previous chapter, software agents are commonly defined as [Wooldridge and Jennings, 1995]: *An agent is an encapsulated computer system that is situated in some environment and that is*

capable of flexible, autonomous action in that environment in order to meet its design objectives.

A few of the notions introduced in this definition are worth further explanation. By *'encapsulated computer system'* is meant that there is a clear distinction between the agent and its environment. Moreover, the definition implies that there is a well-defined boundary and concrete interface between the agent and its environment. The key aspect of the definition is *autonomy*, which refers to the principle that agents can operate on their own without the need for human guidance. An autonomous agent has the control over its own actions and internal state, that is, an agent can decide whether to perform a requested action. The definition situates an agent in a particular *environment*, which the agent can sense and effect. This indicates *responsive behaviour*. Furthermore, the definition implies that agents are *problem solving entities*, with well-defined boundaries and interfaces, designed to fulfil a specific purpose, which is, having particular goals to achieve, and exhibiting flexible and pro-active behaviour.

Agents are often regarded as socio-cognitive entities capable of individual social behaviour [Weber, 1978]. For an agent to be termed cognitive it must be endowed with mental attitudes representing the world and motivating action [Panzarasa *et al.*, 2002], [Wooldridge, 2000]. Further, for a cognitive agent to be deemed socio-cognitive it must not only have an intentional stance towards the environment, but also assume other agents to be cognitive entities similarly endowed with mental attitudes for representational and motivational purposes [Dennet, 1987]. Social behaviour is characterized by the ability to communicate and cooperate with others and with users. Lastly, for agents to be truly intelligent, they must be able to learn as they react and interact with their external environment [Nwana *et al.*, 1999]. Considering these characteristics of agents, and their applications, agents can be classified in different categories, [Franklin and Graesser, 1997]. Agent

taxonomies classify different agent types including software agents, life-like agent (like humans and artificial life types) and robots.

The concept of describing problem solving in terms of agents is becoming more and more popular in a variety of different research disciplines within AI, mainstream computer science and neighbouring disciplines, such as psychology, sociology, economics, etc. [Jennings, 1999, Weiss, 1999]. Already the wide and diverse use of the term agent within common life (e.g. in the sense of travel agent, secret agent, or softening agent), makes it difficult to provide an exact definition of this notion. The common dictionaries provide, in general, several distinctive definitions. For example, Webster's New Encyclopaedic Dictionary [Harkavy, 1996] distinguishes:

- *1a: something that produces or is capable of producing an effect (a cleansing agent);*
- *1b: a chemically, physically, or biologically active principle;*
- *2: one that acts or exerts power;*
- *3: one who acts for or in place of another and by the other's authority (government agents, a real estate agent).*

In general, *1a* and *2* are strongly related because they express the same basic property of an agent from two perspectives. Of course, *'one that acts'* is likely to *'produce an effect'* and normally, the purpose of acting is to produce an effect. Thus, *1a* can be considered as a goal-directed description of definition *2* and both definitions could even be combined, for example, as *'one/something that acts (or exerts power) with the purpose/goal of producing an effect (and possibly the capability of producing this effect)'*. Thus, the basic property of an agent, that can be determined from *1a* and *2*, is *'to act in order to produce an effect'*. In regard to *1b*, from the perspective of computer science, an agent could also be considered as a computational active principle, although, without reference to a definition of what an active principle is

considered to be, such a definition would not be sufficient. *3* is often used in order to describe agents in the contexts of personal assistants [Maes, 1994], [Decker *et al.*, 1997]. Such agents, for example, act as email filterers [Lashkari *et al.*, 1997], [Maes, 1997], meeting schedulers [Kautz *et al.*, 1994], [Garrido and Sycara, 1995], [Jennings, 1995] or mobile agents (or softbots), which search through the Internet [Etzioni and Weld, 1994], [Wayner, 1995a], [Wayner, 1995b]. They are supposed to act on behalf of and by the user's authority. However, such personal assistants are only one of the many different kinds of agents used within the scientific community. Therefore, *3* does not add any commonly agreed property of an agent besides the basic property of acting.

Due to the multi-disciplinary interest in the agent concept, it is also difficult to provide a sound scientific definition [Bond and Gasser, 1988], [Franklin and Graesser, 1997] and until now researchers were not able to agree upon a universal consensus [Jennings and Wooldridge, 1999]. However, recently Russell and Norvig's definition:

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." [Russell *et al.*, 1995] establishes itself as general but widely accepted and used definition because it concentrates on the most basic features of an agent (namely, the representation as an encapsulated entity situated in an environment which perceives and acts upon this environment). This definition provides the basic agent skeleton with the minimum necessary conditions for agenthood (see *figure 11*).

Additionally, it supplies two black boxes representing the internal structure of the agent and the environment that the agent is situated in. Any controversially discussed features and properties of particular agents (such as autonomy, intelligence and rationality) and particular demands on the environment (such as being of physical nature) are explicitly excluded from

the general definition of agency/agenthood. They can be additionally introduced, explained and added (or explicitly excluded) as appropriate. For example, autonomy is an attribute often quoted to be a necessary requirement for agents [Wooldridge and Jennings, 1995], [Nwana and Ndumu, 1998], [Huhns and Singh, 1998], [Sycara, 1998a], whereas mobility is a property needed only for very specific domains, for example, to search through the Internet [Wayner, 1995b], [Wayner, 1995a].

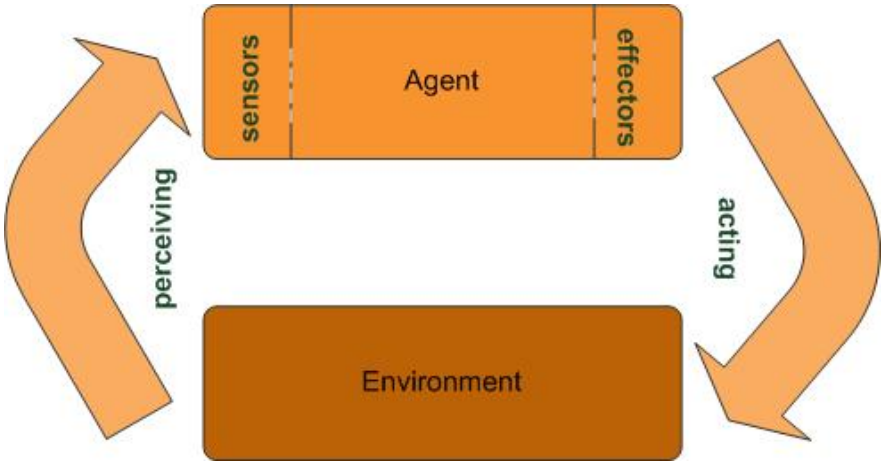


Figure 11. *Agent skeleton.*

Concentrating on the presented essentials allows the consideration of human agents, as well as artificial agents (both software agents and robotic agents) and, therefore, enables a broad scope of agent research within a variety of research disciplines to be covered. Russell and Norvig's definition also by-passes the formidable question and lengthy discussion on what an agent is and what makes it distinct, for example, from any software program (for a discussion without a sufficient answer, the interested reader is referred to [Franklin and Graesser, 1997]). This definition provides the basics for the pragmatic answer, adopted from Shoham, that what makes any entity an agent is precisely the fact that one has chosen to analyse it with this concept

[Shoham *et al.*, 1997]. Thus, if something can and is represented as an agent in the sense of Russell and Norvig, then it is an agent.

3.1.1. AGENT ATTRIBUTES

As mentioned previously, Russell and Norvig's definition does not include any properties or attributes associated with the agent metaphor which are not universally agreed. However, for any branch of research that is working with the agent concept, this definition can be considered as at least a necessary, if not as a sufficient, description of agenthood. Depending on the main purpose for which the agents are constructed, particular attributes need to be added for a useful agent definition. For the demands of this thesis, the key attributes that will be focused upon are those associated with intelligent agents in terms of DAI research. For DAI research, in general, an intelligent agent is a software computer system with the following attributes [Wooldridge and Jennings, 1995, Jennings and Wooldridge, 1999, Sycara, 1998a], that will be explained next: *situatedness*, *autonomy*, *adaptability* and for the case of the intelligent agent being situated within a multi-agent system *sociability*.

3.1.1.1. SITUATEDNESS

Considering Russell and Norvig's definition, roughly speaking, anything that can be viewed as obtaining an input and producing an output, can be viewed as an agent. To this extent, any function or any kind of software can be considered an agent.

However, this consideration neglects to emphasise an important characteristic that constitutes agenthood, and that is included in Russell and Norvig's definition, namely, the *situatedness* of the agent within an environment [Jennings *et al.*, 1998, Sycara, 1998a]. The emphasis that an agent can be viewed as an encapsulated entity

situated in an environment that interacts with the environment only via its sensors and effectors is the reason for the widespread acceptance of Russell and Norvig's definition as description of a standard agent [Wooldridge, 1999].

3.1.1.2. AUTONOMY

Besides situatedness, *autonomy* is the second crucial property which provides the underlying power of the agent paradigm. There exist many slightly different definitions of what constitutes an autonomous agent [Castelfranchi, 1990, Russell *et al.*, 1995, Wooldridge and Jennings, 1995].

For example, Huhns and Singh identify five different varieties of autonomy, which serve different purposes in the study and design of agents [Huhns and Singh, 1998]. However, for the remainder of this thesis, the following description is sufficient: autonomy means *"that agents are able to act without the intervention of humans or other systems: they have control both over their own internal state and over their behaviour"*. [Wooldridge, 1999].

Therefore, an agent is autonomous to the extent that its behaviour depends on its own situational experience at run-time (i.e. its own perceptions of the environment), rather than on built-in knowledge of the environment initially provided by the agent's designer at design-time. So, the agent lacks autonomy if it does not need to pay attention to its possible perceptions because its action choices are determined solely by the designer's built-in knowledge [Russell *et al.*, 1995].

To illustrate that autonomy is a crucial characteristic of intelligent agents, consider the example of an agent that would permanently act blindly (i.e. regardless of the possible perceptions

from the environment) and still always perform successful actions. Besides the fact that such an agent would not be very successful as soon as the environment changes in an unexpected manner, the intelligence behind its apparently intelligent behaviour must be credited solely to the agent's designer who would have been able to predict the best possible actions for all possible situations in advance. Therefore, an intelligent agent needs at least a small degree of autonomy to justify that the intelligence is credited to the agent.

However, an autonomous agent does not need to be intelligent. For example, any monitoring process control system (ranking from simple thermostats to complex nuclear reactor control systems) and any software daemon (such as the UNIX xbiff email-program) performs actions on the basis of the perception of its environment without direct human intervention, and therefore, can be considered as an autonomous agent [Wooldridge and Jennings, 1995, Jennings and Wooldridge, 1999].

Nevertheless, these autonomous agents are typically not considered intelligent agents because they are designed to perform clearly-specified actions within a specific problem domain, whereas *"an intelligent agent is a computer system that is capable of flexible autonomous action in order to meet its design objectives"* [Jennings and Wooldridge, 1999] and moreover *"a truly autonomous intelligent agent should be able to operate successfully in a wide variety of environments, given sufficient time to adapt"*. [Russell *et al.*, 1995]. Thus, autonomy is a necessary prerequisite of intelligent agenthood, but for a sufficient characteristic of intelligent agenthood, additional attributes such as flexibility and, more generally, adaptability need to be addressed.

3.1.1.3. FLEXIBILITY AND ADAPTABILITY

From an AI standpoint of intelligent agents, *flexibility* requires two, to some extent opposing, properties: *responsiveness* and *pro-activeness* [Wooldridge and Jennings, 1995, Jennings and Wooldridge, 1996, Jennings and Wooldridge, 1999]. In this context, responsiveness is defined as the property that *"agents should perceive their environment (which may be the physical world, a user, a collection of agents, the Internet, etc.) and respond in a timely fashion to changes that occur in it"* [Jennings and Wooldridge, 1996]. Whereas *pro-activness* means that *"agents should not simply act in response to their environment, they are able to exhibit opportunistic, goal-directed behaviour and take the initiative where appropriate"* [Jennings and Wooldridge, 1996].

Then, flexibility is obtained by an effective balance between reactive and goal-directed behaviour. However, a good balance is hard to obtain (even for humans) and provides the essence of intelligent behaviour [Wooldridge, 1999].

In general, if possible, an intelligent agent should try to achieve its goals in a systematic long-term manner, which may involve complex procedure-like patterns of actions. However, if necessary, such an agent should be able to react within an appropriate time-scale to present changes in its environment which necessitate changing, postponing, or dropping the currently envisaged goal-achievement. So, the difficulty is to decide when it is best to keep focussed on a goal long enough to eventually achieve it, and when it is better to react differently because the current circumstances make it necessary to adapt immediately to the new situation.

For a truly autonomous intelligent agent, the knowledge about how to balance reactive and goal-directed behaviour should not be entirely specified as initial built-in knowledge at design-time but at run-time obtained from the environment and the agent's own experience [Russell *et al.*, 1995, Nwana and Ndumu, 1998, Sycara, 1998a].

Therefore, some scientists do not only assume flexibility to be an essential requirement of intelligent agents but additionally, the ability to learn from its own experience and its environment [Nwana and Ndumu, 1998, Sycara, 1998a]. Following this view, Sycara extended Jennings and Wooldridge's widely used list of key characteristics for intelligent agents [Wooldridge and Jennings, 1995, Jennings and Wooldridge, 1999]. She determined *situatedness*, *autonomy*, and *adaptability* as the main characteristics of intelligent agents and identified three basic requirements for *adaptability* [Sycara, 1998a]: *responsiveness*, *pro-activeness*, and *the ability to learn*.

It is assumed that these attributes uniquely characterise an intelligent agent. So, when a single software entity possesses these attributes, it can be considered an intelligent agent. However, these properties are not independent of each other.

For example, to be able to adapt to the environment, an agent needs to be able to behave in a flexible manner. However, a lack of autonomy implies a lack of flexibility, because no possibility exists to react to unexpected changes in the environment [Russell *et al.*, 1995], and, therefore, a lack of autonomy also implies a lack of adaptability [Sycara, 1998a, b].

3.1.1.4. SOCIABILITY

The aforementioned attributes are sufficient to characterise an intelligent agent within an agent-based system [Wooldridge and Jennings, 1995]. However, for interacting agents situated within a multiple agent environment, a further property is essential, namely: *sociability*. In this context, sociability means "*that an agent is capable of interacting in a peer-to-peer manner with other agents or humans*." [Sycara, 1998a].

Therefore, "*agents should be able to interact, when they deem appropriate, with other software agents and humans in order to complete their own problem solving and to help others with their activities where appropriate*" [Jennings and Wooldridge, 1996]. Such agents can, for example, interact by *coexistence*, *cooperation*, *negotiation*, or *competition* [Moulin and Chaib-Draa, 1996, Jennings *et al.*, 1998]. In the case of pure *coexistence*, interactions take place indirectly through the environment, for example, by performing actions that change the environment so that other agents may become affected, or by observing one another [Weiss, 1999]. However, for most high-level forms of interaction, such as *cooperation* and *negotiation*, interaction can also take place directly, for example, by communication through a shared agent-communication language [Genesereth and Ketchpel, 1994, Jennings and Wooldridge, 1995]. To engage in an intelligent manner in sophisticated patterns of interaction, the agents must not only be able to follow simple communication strategies such as information exchanges and requests for particular actions to be performed, but the agents must be able to participate and follow complex communication, negotiation and other interaction protocols [Huhns and Singh, 1998].

Therefore, the sociability attribute implies that intelligent agents situated within a multi-agent system need at least the following requirements to interact in an intelligent manner: the ability to become aware of the possible co-existence of other agents, a possibility to represent and reason about each other (for example, in terms of the other agents' knowledge, goals, plans, and possible actions), and facilities to communicate with one another in an appropriate manner [Bond and Gasser, 1988, Huhns and Singh, 1998].

As it is the case with some of the others, the sociability attribute is not independent of the other key properties of intelligent agents. In principle, from a technical standpoint, sociability does not even need to be added as an extra property of the character of an intelligent agent, but it can be incorporated in the other properties. Firstly, the other agents are part of the overall environment of an agent, and therefore, any interactions with the other agents can only happen by performing actions (which is already addressed by the situatedness aspect). For example, to communicate with other agents in the environment, the agent needs to perform some form of communicative actions, such as speech acts [Austin, 1962, Searle, 1969, Genesereth and Ketchpel, 1994]. Secondly, by assuming that the other agents might be acting autonomously, the environment may be changed in a flexibly way by actions caused by the other agents, and therefore, an agent should be able to react flexible to environmental changes caused by the other agents and, ultimately, an intelligent agent should be able to adapt to (and influence) the behaviour of the others [Jennings *et al.*, 1998, Sycara, 1998a, Castelfranchi, 1998].

Because the sociability attribute only becomes important in the context of multiple agent environments, it is legitimate to address it as an additional key property of intelligent agents, although it can be

entirely incorporated into the other key aspects of intelligent agenthood. However, sociability is the central focus of research in intelligent agents from the DAI perspective.

3.1.2. AGENT ARCHITECTURES

Concerning the implementation of agents, several architectures have been proposed that can be roughly classified into the following types [Wooldridge, 1999], increasingly less abstract:

- *Logic-based agents:* reasoning and decision making are realized through logical deduction [Genesereth and Nilsson, 1987, Lesperance *et al.*, 1996, Fisher, 1994].
- *Reactive agents:* in which decision making is implemented as some direct mapping from situation to action [Brooks, 1986, Maes, 1990].
- *Belief-desire-intention (BDI) agents:* decision making depends on the manipulation of some representation of the beliefs, desires and intentions of the agent [Bratman *et al.*, 1988, Rao and Georgeff, 1992].
- *Layered agents:* decision making is realized via several software layers, each explicitly reasoning about the environment at different levels of abstraction [Müller *et al.*, 1995, Ferguson, 1995].

Of the above architectures, special attention will be paid to the *BDI* architecture. On the one hand, this architecture has become a *de facto* standard for agent models and is at the basis of namely the FIPA standard, and, on the other hand, it is generic enough to enable the modelling of both natural as artificial agents. Being a generic architecture, *BDI* provides the best approach to this requirement.

The *BDI* model has its roots in the philosophical tradition of understanding practical reasoning in humans (e.g. [Bratman *et al.*, 1988,

Cohen and Levesque, 1990]). Practical reasoning involves two important processes: deciding what goals to achieve (deliberation), and how to achieve those goals (means-ends analysis). The process starts by analyzing the options available, which depend on the agent's beliefs and desires, and deciding which ones to choose.

These chosen options became the agent's intentions, which then determine its actions. Intentions play a crucial role in the practical reasoning process, as they lead to action. Important aspects of intentions are [Bratman, 1987, Wooldridge, 2000]:

- *Lead the means-ends reasoning process:* once an intention is formed, the attempt to achieve it involves deciding how.
- *Constrain future deliberation:* a rational agent will not entertain options that are inconsistent with its intentions.
- *Persistency:* agents will not give up their intentions without a good reason. Intentions persist until they are achieved or found impossible to achieve.
- *Influence beliefs:* Plans for the future will be based in the belief that the intentions will be achieved.

In summary, agents have a set of *beliefs*, which are based on their perception of the environment. *Beliefs* and *intentions* are used to determine the current options (*desires*) available to the agent. A deliberation process determines the agent's *intentions* based on its *beliefs*, desires and *intentions*. *Intentions* are the current focus of the agent: the states it is committed to bring about, and for which the agent will specify a plan on how to reach them.

Finally, an action selection function, determines which action to perform based on the current intentions. This process of practical reasoning in a *BDI* agent is described in *figure 12.*

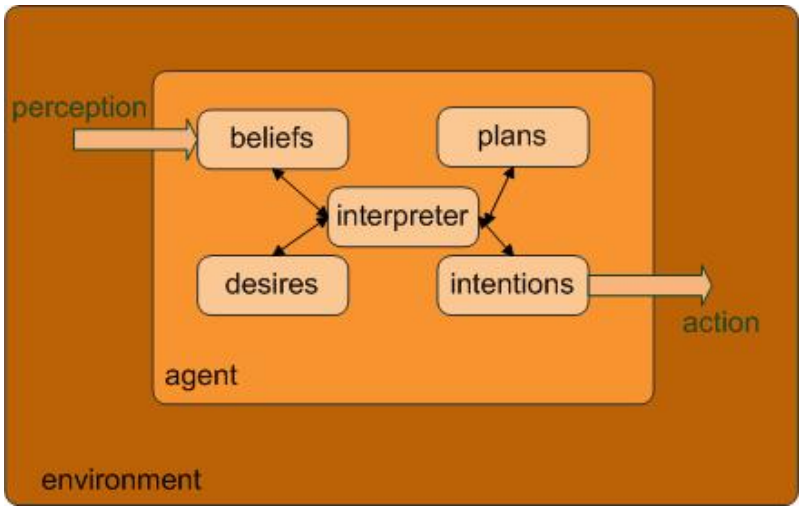


Figure 12. *The BDI agent model.*

BDI models have been applied to a number of practical problems including air traffic control, spacecraft handling and telecommunications management and a great deal of effort has been devoted to their formalization [Rao and Georgeff, 1992]. The best known implementation of the *BDI* model is the PRS system [Georgeff and Lansky, 1987]. Finally, *BDI* models have been extended by many researchers, for example to include communication between agents [Haddadi, 1996, Dignum *et al.*, 2000], or normative behaviour [Broersen *et al.*, 2001].

3.1.3. APPLICABILITY OF AGENTS

Having briefly introduced agents and their characteristics, it is important now to describe in which cases the agent paradigm can or should be used. That is, what do agents have to offer? According to [Jennings and Wooldridge, 1998] the usefulness of any technology should be judged in two directions: first, its ability of solving new types of problems, and second its ability to improve the efficiency of current solutions.

The agent paradigm provides a natural way to view and characterize intelligent and/or reactive systems [Weiss, 1999]. Intelligence and interaction are deeply and inevitably coupled, and multi-agent systems reflect this insight. Multi-agent systems can provide insights and understanding about poorly understood interactions between natural, intelligent beings, as they organize themselves into groups, societies and economies in order to achieve improvement.

Systems that maintain an ongoing interaction with some environment are inherently quite difficult to design and correctly implement. Process control systems and network management systems are examples of such reactive systems. Applications of the agent paradigm can be broadly divided in three classes: open systems, complex systems and ubiquitous systems.

Open systems are systems in which the structure of the system is capable of dynamically changing. Their components are not known in advance, can change overtime, and may be highly heterogeneous. An excellent example of an open system is the Internet. Any computer system that must operate in the Internet must be capable of dealing with many and very different organizations and agendas, without constant guidance from users. Such functionality is almost certain to require techniques based on negotiation and co-operation, which lie firmly in the domain of multi-agent systems.

Complex systems relate to particularly complex, large or unpredictable domains. The most powerful tools to deal with complexity in systems are modularity and abstraction. Application of the agent paradigm entails that the overall problem can be partitioned into a number of sub-problems of less complexity that are easier to handle. This decomposition allows agents to employ the most appropriate paradigm to solve a sub-problem. The notion of an autonomous agent is also a

powerful abstraction, in just the same way as data types or objects.

Ubiquitous systems have the goal of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user. Ubiquitous systems are roughly the opposite of virtual reality. Where virtual reality puts people inside a computer-generated world, ubiquitous computing forces the computer to live out there in the world with people [Weiser, 1993]. In ubiquitous systems the need for an equal partnership between the system and its user is paramount. The system has to cooperate with the user to reach their goal. It has been predicted that in the future, delegating to, rather than manipulating computers [Negroponte, 1996] will drive computing. Software applications to deliver such functionality need to be autonomous, pro-active, responsive and adaptive. In other words, such applications need to behave as an intelligent agent. This gives rise to the idea of *'expert assistants'*, which are agents knowledgeable about both the application and the user.

Agent technology has been successfully applied to several of the above types of systems. However, the fact that a system can be designed as a (multi-)agent system does not mean that an agent-based solution is always the most appropriate one. Other pitfalls to the development of agent-based systems have been discussed in [Wooldridge and Jennings, 1999].

These include political (overselling agents), management (using agents no matter what), conceptual (the risk of the silver bullet), and development (yet another agent architecture) pitfalls. From a software engineering perspective, there are basically four limitations to the use of agents [Jennings and Wooldridge, 1998]:

- *Agent systems have no overall system controller.* An agent-based solution may thus not be appropriate in situations where global

constraints have to be maintained.

- *Agents have local perspective.* Agent actions are determined by its own local state. Since in most applications, agents do not maintain complete global knowledge, this may mean that agents make global sub-optimal decisions. One of the aims of multi-agent systems research is to reconcile decision making based on local knowledge with the desire to achieve globally optimal performance [Bond and Gasser, 1988].
- *Trust and delegation limitations.* Both individuals and organizations have to be confident that agents will work on their behalf. The process of learning to trust an agent and to learn how to delegate tasks to an agent takes time.
- *Careful personalization limitations.* Profiles that an agent makes of its user must be comprehensive, accurate, require minimal user input, and enforce privacy issues. Furthermore an agent must know its limitations and be trustworthy.

3.2. MULTIAGENT SYSTEMS

Multi-agent environments extend single-agent architectures with an infrastructure for interaction and communication. Ideally, MAS exhibit the following characteristics [Huhns and Stephens, 1999]:

- Are typically *open* and have *no centralized designer*.
- Contain *autonomous, heterogeneous* and *distributed* agents, with different 'personalities' (cooperative, selfish, honest, etc.).
- Provide an infrastructure to specify *communication* and *interaction* protocols.

Agents in a MAS are expected to coordinate by exchanging services and information, to be able to negotiate and agree on commitments, and to

perform other complex social operations. *Coordination* and *communication* are therefore extremely important issues of MAS, but not really relevant in the case of single-agent systems. In MAS agents have to be able to find each other, announce their possibilities and pose questions or requests. Furthermore, MAS infrastructure must provide security services, to ensure that agents do not misbehave.

Several architectures and models for MAS have been proposed that handle coordination in different ways. One of the initial and most widely used architectures is based on mediators. The concept of mediator was first introduced by Gio Wiederhold [Wiederhold, 1992] as a way to deal with the integration of knowledge from heterogeneous sources. Mediators are facilitation agents that can provide a number of intermediate information services to other agents. They may suggest collaboration between users with common interests, or provide information about tools and resources available. An example of a MAS infrastructure based on the concept of mediators is RETSINA [Sycara *et al.*, 2003]. RETSINA was implemented based on the idea that agents in the system form a community of peers that engage in peer to peer relations. Coordination should emerge from the relations between agents rather than be imposed by the infrastructure, and as such does not employ centralized control but provides (mediation) services that facilitate the relations between agents.

3.2.1. AGENT SOCIETIES

The term *society* is used in a similar way in agent societies research as in human or ecological societies. The role of any society is to *allow its members to coexist in a shared environment and pursue their respective roles in the presence and/or in cooperation with others.* Main aspects in the definition of society are *purpose*, *structure*, *rules* and *norms*. *Structure* is determined by *roles*, interaction *rules* and communication *language*.

Rules and *norms* describe the desirable behaviour of members and are established and enforced by institutions that often have a legal standing and thus lend legitimacy and security to members. A further advantage of the organization-oriented view on designing multi agent systems is that it allows for heterogeneity of languages, applications and architectures during implementation.

Organizations can be seen as sets of entities regulated by mechanisms of social order and created by more or less autonomous actors to achieve common goals. Multi-agent systems that model and support organizations should therefore be based on coordination frameworks that mimic the structure of the particular organization and be able to dynamically adapt to changes in organization structure, aims and interactions. The structure of the organization determines important autonomous activities that must be explicitly organized into autonomous entities and relationships in the conceptual model of the agent society [DignumWeigand *et al.*, 2002].

In a business environment, the behaviour of the global system and the collective aspects of the domain –such as stability over time, predictability and commitment to overall aims and strategies– must be considered. That is, the concept of desirable social behaviour is of utmost importance when multi-agent systems are considered from an organizational point of view. This leads to a rising awareness that multi-agent systems and cyber-societies can best be understood and developed if they are inspired by human social phenomena [Artikis *et al.*, 2001, Castelfranchi, 2000, Zambonelli *et al.*, 2001]. This is, in many ways, a novel concept within agent research, even if sociability has always been considered an important characteristic of agents.

When multi-agent systems are considered from an organizational point of view, the concept of desirable social behaviour becomes of

utmost importance. That is, from the organizational point of view, the behaviour of individual agents in a society should be understood and described in relation to the social structure and overall objectives of the society. Until recently, multi-agent systems were mainly viewed from an individualistic perspective, that is, as aggregations of agents that interact with each other, and how an agent can affect the environment or be affected by it [Ferber and Gutknecht, 1998]. This view looks at the behaviour of multi-agent systems from the perspective of the agent itself, in terms of how an agent can affect the environment or be affected by it.

The term agent society will be used to refer to MAS considered from a social perspective. In an individualistic view of Multi-Agent Systems, agents are individual entities *socially situated* in an environment, that is, their behaviour depends on and reacts to the environment, and to other agents on it [Dautenhahn, 2000]. It is not possible to impose requirements and objectives to the global aspects of the system, which is paramount in business environments. However, organization-oriented agent societies require a collectivist view on the relation between agent and environment. That is, agents are considered as being *socially embedded* [Edmonds, 1999]. If an agent is socially embedded it needs to consider not only its own behaviour but also the behaviour of the system as a whole and how agents in the system influence each other. Davidsson has proposed a classification for artificial societies based on the following characteristics [Davidsson, 2001]:

- *Openness,* describing the possibilities for any agent to join the society.
- *Flexibility*, indicating the degree agent behaviour is restricted by society rules and norms.
- *Stability,* defining the predictability of the consequences of actions.

– *Trustfulness*, specifying the extent to which agent owners may trust the society.

Depending on its purpose, a society needs to support these characteristics in different degrees. In one extreme, there are *open societies* that impose no restrictions on agents joining the society. Popper has defined open societies as systems in a state, far from equilibrium, that shows no tendency towards an increase in disorder [Popper, 1982]. That is, open societies support flexibility and openness very well but lack on stability and trustfulness. The most obvious example of an open society is the WWW. Open agent societies assume that participating agents are designed and developed outside the scope and design of the society itself and therefore the society cannot rely on the embedding of organizational and normative elements in the intentions, desires and beliefs of participating agents but must represent these elements explicitly. These considerations lead to the following requirements for engineering methodologies for open agent societies [Dignum and Dignum, 2001].

Agent societies must include formalisms for the description, construction and control of the organizational and normative elements of a society (roles, norms and goals) instead of just the agents' states [Artikis *et al.*, 2001, Zambonelli *et al.*, 2001].

The methodology must provide mechanisms to describe the environment of the society and the interactions between agents and the society, and to formalize the expected outcome of roles in order to verify the overall animation of the society.

The organizational and normative elements of a society must be explicitly specified since an open society cannot rely on its embedding in the intentions, desires and beliefs of each agent [Dellarocas and Klein, 2000b, Ossowski, 1999].

Methods and tools are needed to verify whether the design of an agent society satisfies its design requirements and objectives [Jonker *et al.*, 2000].

The methodology should provide building directives concerning the communication capability and ability to conform to the expected role behaviour of agents participating in the society.

In *closed societies***,** on the other extreme, it is not possible for external agents to join the society. Agents in closed societies are explicitly designed to cooperate towards a common goal and are often implemented together with the society [Zambonelli *et al.*, 2001]. Closed societies provide strong support for stability and trustfulness properties, but only allow for very little flexibility and openness. The large majority of existing MAS are closed.

[Davidsson, 2001] introduces two new types of agent societies, semi-open and semi-closed, that combine the flexibility of open agent societies with the stability of closed societies. This balance between flexibility and stability results in systems where trust is achieved by mechanisms that enforce ethical behaviour between agents

In *semi-open societies* the access of external agents is explicitly regulated. This allows deciding on the acceptance or not of new members and to monitor which agents are currently in the society. An example of a semi-open society is the Napster systemT5T. Semi-open societies slightly limit the openness and flexibility characteristics of open societies, but are able to provide greater stability and trustfulness.

Semi-closed societies do not allow for the participation of external agents but provide the possibility for external parties to initiate a new agent within the society to act on their behalf. This extends the flexibility and openness of the society, without losing on stability and trustfulness, since participating agents are still designed following the society

requirements and the owner of the society still controls the overall architecture of the system. Semi-closed societies are as open as semi-open society but less flexible. This is the approach taken in the ISLANDER platform where external agents are provided with an API as interface to the institution, which regulates and controls all interaction [Esteva *et al.*, 2002].

3.2.2. COORDINATION IN MULTIAGENT SYSTEMS

Multi-agent systems that are developed to model and support organizations need coordination frameworks that mimic the coordination structures of the particular organization. The organizational structure determines important autonomous activities that must be explicitly organized into autonomous entities and relationships in the conceptual model of the agent society [DignumWeigand *et al.*, 2002]. Furthermore, the multi-agent system must be able to dynamically adapt to changes in organization structure, aims and interactions.

Coordination can be defined as *the process of managing dependencies between activities* [Malone and Crowston, 1994]. Organizational science and economics have since long researched coordination and organizational structures [Williamson, 1975, Powell, 1991]. Drawing on disciplines such as sociology and psychology, research in organization theory focuses on how people coordinate their activities in formal organizations. On the other hand, it is also generally recognized that coordination is an important problem inherent to the design and implementation of multi-agent systems [Bond and Gasser, 1988].

The challenge of coordination in MAS has been recognized by many authors and several approaches have been developed and advocated. Such approaches take either a bottom-up (e.g. goal management in which members of the group take control of the definition of their work [Malone

and Crowston, 1994]) or a top-down view of coordination (e.g. shared ontologies [Fox and Gruninger, 1998] and the hierarchical assignment of responsibilities used in many human organizations). Coordination is one of the cornerstones of agent societies and is considered an important problem inherent to the design and implementation of MAS [Bond and Gasser, 1988, Dignum and Dignum, 2001]. However, the implications of coordination models to the architecture and design of agent societies are not often considered. Other examples of coordination theories in MAS are joint-intentions [Cohen and Levesque, 1990, Dunin-Keplicz and Verbrugge, 2002], shared plans [Grosz and Kraus, 1996] and domain-independent teamwork models [Tambe, 1997].

Behavioural approaches to the design of multi-agent systems are gaining terrain in agent research and several research groups have presented different kind of models. Recent developments recognize that the modelling of interaction in MAS cannot simply rely on the agent's own (communicative) capabilities. Furthermore, organizational engineering of MAS cannot assume that participating agents will act according to the needs and expectations of the system design. Concepts as organizational rules [Zambonelli, 2002], norms and institutions [Esteva *et al.*, 2001] and social structures [Parunak and Odell, 2002] all start from the idea that the effective engineering of MAS needs high-level, agent-independent concepts and abstractions that explicitly define the organization in which agents live [Zambonelli *et al.*, 2001].

Relating society models to the organizational perception of the domain can facilitate the development of organization-oriented multi-agent systems. This means that the development of agent society models for organizations must be a concerted effort between MAS engineers and domain specialists. A common ground of understanding is therefore needed between MAS engineers and organizational practitioners.

Coordination aspects are relevant both in agent research as in organizational theory. Therefore, coordination is considered the way to bridge both communities and create an initial common ground for cooperation.

3.2.3. COMMUNICATION

The main challenge of coordination and collaboration among heterogeneous and autonomous intelligent systems (taking into account both humans and software) in an open, information-rich environment is that of mutual understanding. Only by sharing a mutual understanding of the domain will agents be able to exchange and combine information from heterogeneous sources. Communication and social interaction are therefore the core characteristics of autonomous agents. A mechanism for communication must include both a knowledge representation language (to specify the internal behaviour of agents) and a communication protocol (to specify the interactions among agents). Knowledge representation models are based on *ontologies* that define the domain model and vocabulary of a particular domain of discourse, and shared using *content languages* that represent the agent's mental model of the world (e.g. beliefs, desires, and intentions). Given a particular domain of discourse and a particular community of agents that know and do something in this domain, a communication language is needed that can model the flow of knowledge and attitudes about such knowledge within the agent community. In the following communication protocols and knowledge representation languages are described in more detail.

An *Agent Communication Language* (ACL) provides language primitives that implement the agent communication model. ACLs are commonly thought of as *wrapper languages* in that they implement a knowledge-level communication protocol that is unaware of the choice of

content language and ontology specification mechanism. Most work done in the area of agent communication languages is based on the *Language Action Perspective* [Winograd, 1987] and *Speech Act Theory* [Searle, 1969], a formal model of human communication developed by philosophers and linguists.

Speech Act Theory [Austin, 1962, Searle, 1969] sees human natural language as actions, such as requests, suggestions, commitments and replies. *Speech Act theory* states that a language is used not only for making a statement but it also performs actions. For example, when someone asks someone else to do something, he/she is already causing an action. In *Speech Act Theory*, organizational communication is seen as the exchange of speech acts for the purpose of coordinating organizational activities. The theory provides the means to analyze communication in detail at three levels: *content* (locution), *intention* (illocution) and *effect* (perlocution). *Locution* is the information contained in an utterance. *Illocution* is the purpose that an utterance has, like informing, convincing, requesting, or demanding. *Perlocution* is the actual effect that a statement has. Form (syntax) of communication is less important than *'why'* and *'what'* is communicated.

Speech Act Theory is relevant to agent communication in that it serves as one (but not the only) formal basis for deciding on agent communication language primitives. Using speech act theory eases ambiguous semantic resolution, as compared to the natural languages. Speech acts are useful in that one can formally represent the intent of the speaker and the effect on the hearer. It is up to the agent theory and the agent infrastructure to ensure that agents in the community are ethical and trustworthy, and therefore that the perlocutionary behaviour of a speech act on the hearing agent is predictable. All this is not the concern of ACLs, which are merely providing the language primitives. Still, the semantics of

speech acts for a particular agent completely depends on the agent's belief, intention, knowledge about how to carry out the operation, and the society to whom an agent belongs. These semantics are represented using the knowledge representation language. *The Language Action Perspective* (LAP) is a practical application of the *Speech Act Theory*, which is used as a linguistic tool to model communication in *Cooperative Information Systems* [Flores and Ludlow, 1976]. The basic assumptions underlying the *Language Action Perspective* are [Verharen, 1997].

The primary dimension of human cooperative activity is language. Action is performed through language in a world constituted by language. The meaning of sentences for the actors in a social setting is revealed by the kinds of acts performed. Cooperative work is coordinated through language acts. The speech act is the basic unit of communication. Speech acts obey socially determined rules.

The design of IT systems has a focus on getting things done, whenever work involves communication and coordination among people. The act of doing something, the patterns of interaction and their articulation are the primary concern of information systems design.

Recent developments in the area of agent communication have resulted in the definition of two different ACLs based on the Speech Act Theory. The first one is KQML (Knowledge Query and Manipulation Language) developed in the context of the ARPA Knowledge Sharing Effort [Finin et al., 1994]. KQML consists of a set of communication primitives (called performatives, in accordance to Speech Act Theory terminology) which aim to support cooperation among agents in distributed applications. The KQML performatives enable agents to exchange and request knowledge, and to cooperate during problem solving. KQML doesn't care about the content language used to represent the mental. Its goal is to provide knowledge transportation protocol for

blobs of content, in some ontology that the sending agent can point to and the receiving agent can access.

The second language is FIPA-ACL, the Agent Communication Language framework proposed by the Foundation for Intelligent Physical Agents [Fipa, 2002]. FIPA ACL is associated with FIPA's open agent architecture. As with KQML, FIPA-ACL is based on Speech Act Theory and is independent from the content language and is designed to work with any content language and any ontology specification approach. Furthermore, FIPA-ACL limits itself to primitives that are used in communications between agent pairs. The FIPA architecture has an Agent Management System that specifies services that manage agent communities.

Both FIPA-ACL and KQML are languages similar to those in the family of so-called coordination languages [Carriero and Gelernter, 1992]. These extend sequential languages with constructs to support concurrency and coordination. In a similar way, FIPA-ACL and KQML extend knowledge representation formalisms with knowledge communication primitives, and focus on defining knowledge level coordination languages, which can be used to specify a range of cooperation strategies. Knowledge level coordination languages are situated at a higher level of abstraction with respect 'normal' coordination languages of distributed computing, as they support coordination not at the symbol-level but at the knowledge-level [Newell, 1994].

3.3. SUMMARY AND CONCLUSIONS

In this chapter, the multi-agent systems have been described. First, a definition of agent have been exposed, indicating the agent attributes, the existing agent architectures and an approximation to the environments where

agents may be used. The main described characteristics of the agents are: s*ituatedness, autonomy, flexibility and sociability.*

Then, after describing the characteristics of the agents, the multiagent systems are specified. First agent societies are presented, and then the coordination in multiagent systems and the communication required to work correctly, are illustrated.

Agents represent the simple element, the basic element of the structure of the architecture presented in this document. Agents are structured into organizations, which will be explained next. Agents work together to achieve common objectives and allow the architecture to be flexible and fast, to respond to different requests at the same time, without wondering what kind of request it is necessary to respond at a time.

In this chapter, agents and multi-agent systems have showed their capabilities and how they could represent a useful methodology to designing an architecture as the one presented in this document.

In the next chapter, the organizations of agents will be explained. Organizations assume the advantages of agents and multiagent systems, but introduce and organizational point of view in the set of agents implied. Organizations enrich the multiagent point of view, and introduce, at the same time, a big amount of flexibility, in order to be applied to different situations, just by changing the way the agents are organised. The organizations of agents paradigm is the methodology that has been chosen to design the OBaMADE architecture presented here.

”Science is nothing but trained and organized common sense.” *Thomas H. Huxley*

4. Organizations of Agents

The organizational design employed by an agent system can have a significant, quantitative effect on its performance characteristics. A range of organizational strategies have emerged from this line of research, each with different strengths and weaknesses. In this chapter the organizations of agents are introduced describing the concept of organization and the main factors of the organizations.

Organizations represent a pass forward in agents' evolution. The agent paradigm has evolved from an individualization of the work to the coordination of small entities produced in multiagent systems. Those systems considered the collaboration between agents in order to obtain a general, global, common objective, by dividing the work to do in separated pieces that can collaborate. Organizations establish an inner structure within the group of agents and determine different kind of relationships, depending on the way the agents are organized or depending on the goal they should accomplish.

The organization of a multi-agent system is the collection of roles, relationships, and authority structures which govern its behaviour. All multi-agent systems possess some or all of these characteristics and therefore all have some form of organization, although it may be implicit and informal. Just as with human organizations, such agent organizations guide how the members of the population interact with one another, not necessarily on a moment-by-moment basis, but over the potentially long-term course of a particular goal or set of goals. This guidance might influence authority relationships, data flow, resource allocation, coordination patterns or any number of other system characteristics [Hayden *et al.*, 1999, Carley and Gasser, 1999]. This can help groups of simple agents exhibit complex behaviours and help sophisticated agents reduce the complexity of their reasoning. Implicit in this concept is the assumption that the organization serves some purpose – which the shape, size and characteristics of the organizational structure can affect the behaviour of the system [Galbraith, 1974].

It has been repeatedly shown that the organization of a system can have significant impact on its short and long-term performance [Carley and Gasser, 1999, Sandholm *et al.*, 1999, Durfee *et al.*, 1987, Horling *et al.*, 2004, Matson *et al.*, 2003, So and Durfee, 1996, Brooks and Durfee, 2003], dependent on the characteristics of the agent population, scenario goals and surrounding environment. Because of this, the study of organizational characteristics, generally known as computational organization theory, has received much attention by multi-agent researchers.

It is generally agreed that there is no single type of organization that is suitable for all situations [Ishida *et al.*, 1992, Corkill and Lander, 1998, Lesser, 1998, Carley and Gasser, 1999]. In some cases, no single organizational style is appropriate for a particular situation, and a number of different, concurrently operating organizational structures are needed [Gasser, 1991,

Horling *et al.*, 2003]. Some researchers go so far as to say no perfect organization exists for any situation, due the inevitable tradeoffs that must be made and the uncertainty, lack of global coherence and dynamism present in any realistic population [Romelaer, 2002].

What is clear is that all approaches have different characteristics which may be more suitable for some problems and less suitable for others. Organizations can be used to limit the scope of interactions, provide strength in numbers, reduce or manage uncertainty, reduce or explicitly increase redundancy or formalize high-level goals which no single agent may be aware of [Lesser and Corkill, 1981, Fox, 1981].

At the same time, organizations can also adversely affect computational or communication overhead, reduce overall flexibility or reactivity, and add an additional layer of complexity to the system [Horling *et al.*, 2004]. By discovering and evaluating these characteristics, and then encoding them using an explicit representation [Fox *et al.*, 1998], one can facilitate the process of organizational-self design [Corkill and Lesser, 1983] whereby a system automates the process of selecting and adapting an appropriate organization dynamically [Lesser, 1998, Schwaninger *et al.*, 2000]. This approach will ultimately enable suitably equipped agent populations to organize themselves, eliminating at least some of the need to exhaustively determine all possible runtime conditions a priori. Before this can occur, the space of organizational options must be mapped, and their relative benefits and costs understood.

These benefits and costs, and the potential advantages that could be provided by technologies able to make use of such knowledge, motivate the need to determine the characteristics of organizations and under what circumstances they are appropriate. While no two organizational instances are likely to be identical, there are identifiable classes of organizations which share common characteristics [Romelaer, 2002]. Several organizational paradigms suitable for multi-agent systems have emerged from this line of

research [Fox, 1981]. These cover particularly common, useful or interesting structures that can be described in some general form. Several of these paradigms will be described next, giving some insight into how they can be used and generated, and comparing their strengths and weaknesses.

4.1. CONCEPT OF ORGANIZATION

In order to better know how to model organizations in multiagent systems, it is necessary to understand the concepts related with human organizations. Thus, in this sub-section human organizations are first analyzed, and then organizations of agents, will be explained in following sub-sections.

4.1.1.HUMAN ORGANIZATIONS

Human organizations represent the inspiration and clear model to develop any other kind of 'artificial' organizations. This is why human organizations are first explained here and then, and taking this organizations as a model, organizations of agents will be developed.

An organization *"is a social arrangement which pursues collective goals, which controls its own performance, and which has a boundary separating it from its environment"*.

J.M. Peiró defines organization as a *"formation or social entity with a precise number of members and with an inner differentiation of the tasks dealt by every member"* [Peiró, 1995].

I. Guzmán, considers an organization as *"the coordination of the activities of all the individuals that make part of an enterprise with the purpose of obtaining the best possible gain of the material, technical or human means, to achieve the goals of the enterprise"* [Valdivia, 1983].

Another similar definition is proposed by J. Massie, where an organization is a *"cooperative group of human beans where the tasks are assigned among its members and where the relationships are identified and its activities are integrated to achieve common objectives in a structured way"* [Massie, 1973].

Thus, an organization is composed by a series of individuals that make some specific and differentiated tasks or activities. Besides, those individuals are structured following some determined rules that allow them to achieve the objectives of the organization.

The goals should be commonly known, guiding the efforts of the members to be achieved [Peiró, 1995]. The organization should also proportionate a source to legitimate the adequate actions in the organization, establishing the minimum levels or standards to acquire.

A human organization can be characterized by the following characteristics [Hodge *et al.*, 1998]:

- It is *formed by people.*
- *Follows a determined goal,* which guides the activities of the members of the organization, through the coordination and control of the action mechanisms.
- *There is a subdivision of the work among the individuals,* by specialization and division of tasks.
- *Requires a formal structure,* with defined roles (independent of the person that carries that role); responsibilities associated with those roles; and certain previously established relationships between the members of the organization.
- *All the established activities should be related with global objects within the organization.* The existence of certain role is only justified if it is useful to achieve those goals.

– *An organization has defined limits*, establishing the members of the organization (directly naming each member or indicating the situation where the activity takes place).

4.1.2.ORGANIZATIONS OF AGENTS

In the multiagent knowledge field, the term organization has been mainly used to describe a set of agents that, using some kind of roles, interact with each other coordinating themselves to achieve the global objectives of the system.

L. Gasser assumes that organizations are structured systems with activity, knowledge, culture, history, and ability pattern, different of any particular agent [Gasser, 2001]. Organizations exist in a completely different level than individual agents that make up the organizations themselves. Individual agents are replaceable. Organizations are established in a space; it either is geographical, temporal, symbolic, etc. So, an organization of agents proportionates a kind of workspace for the activity and interaction of the agents by defining roles, behavioural expectatives and relations.

F. Zambonelli [Zambonelli *et al.*, 2003] considers the organizations of agents as a set of roles that keep the relationships among them, and that generates interaction patterns with other roles in an institutionalized and systematic way.

Ferber indicates that organizations proportionate a way to divide the system, crating groups or units that form the interaction context of the agents [Ferber *et al.*, 2004]. The organization is then based in two main aspects: structural and dynamic. The structure of the organization represents the remaining components when the individual elements enter or leave the organization. The organization is composed by the set of relationships that allow seeing a number of different elements as unique.

The structure defines the way the agents are grouped in organizational units and how those units are related with each other. The roles needed to develop the activities of the organization are also defined in the structure, as long as the relationships and restrictions.

The organizational dynamics is centred in the interaction patterns defined for the roles, describing the way to get into or to leave the organization, the parameters of the roles and the way the roles are assigned to the agents.

For V. Dignum, the organizations of agents assume that there are global objectives, different from the individual agents' objectives [Dignum and Dignum, 2007b]. Roles represent organizational positions that help to achieve those global objectives. Agents may have their own objectives and decide if they take any specific system role or not, determining which among the available protocols is more suitable to achieve their chosen objectives.

Finally, J. Hubner considers the organizations as a set of behavioural restrictions adopted by a group of agents to control their own autonomy and to help to easier accomplish their global objectives [Hubner *et al.*, 2005].

It is then possible to distinguish an organization of agents by the following characteristics:

- *It is composed by agents* (software, physical or human), independently of their inner characteristics and individual objectives.
- *Follows a global common objective* that does not directly depend on the individual objectives of the particular agents that make part of the organization in every moment.
- *The tasks assigned to the agents are divided in roles,* which describe the activities and functionality of the organization.

- *Organizations proportionate a disaggregation of the system* in groups or units, where the interaction between agents takes place.
- *Organizations have clearly defined limits*, determined by: the organization environment, the internal and external agents and the functionality of the organization and the services offered.

Comparing the resumed characteristics of human and agent organizations, both have quite similar features, motivated by the fact that the organizations of agents are normally developed from the simulation and adaptation of the organizational human behaviours. Thus, is quite reasonable to assume that improving the knowledge of human organizations will help to obtain methods and design guides, as well as new concepts, dimensions and aspects to take into account to analyze, design and implement organizations of agents.

4.2. ORGANIZATION FACTORS

When analysing the organizations, it is important to take into account not only the entities that form the organization, but also their relationships and the objectives they want to achieve. Some other factors are also important when analyzing an organization. They could be: the functionality of the system, the environment where it is place and to which it is related and the behavioural rules that guide the behaviour of their components. Then, the main elements to consider when modelling an organization are the following:

- *Structure:* it is formed by all the elements that remain in the organization independently of the final individual that form the organization in every moment. It is defined by the roles, groups, dependences and relational schemes.
- *Functionality:* specify the main objectives of the organization, the functionalities offered by the organization, the smaller objectives

followed by the different members of the organization and what tasks and plans should be carried out to achieve them.

- *Normalization:* determines the set of rules and actions defined to control the behaviour of the members of the organization. The rules about the way the members should act are also included here (specifying the obligations, prohibitions and permissions of every member, also including penalties and rewards according to their acts).
- *System dynamics:* explains how the organization evolves through the time, indicating the way the agents get into the organization or leave it in a dynamic way. The agents may adopt different roles in according to their capabilities and abilities. Agents make part of those groups of the organization where they are admitted.
- *Environment:* it is formed by the resources to whose the organization depends on; like the providers of those resources, the clients or beneficiaries of the existence of the organization.

Next, all these elements that make part of the organizations will be explained more detailed, paying special attention to the relationship between the agents that form the organizations.

4.2.1. STRUCTURE

In human organizations, the structure of the organization defines how the working tasks are divided, grouped and coordinated. Thus, a key element in the composition of the organizations are the groups [Peiró, 1991], composed by a limited number of individuals that interact with each other and that share a set of values and norms (conduct standards).

The main elements that characterize the structure of an organization are: the specialization, the division into departments, the hierarchy, the control, the centralization and decentralization and the formalization of the

tasks [Hodge et al., 2003].

The specialization or work division indicates the degree of division of the tasks of the organization into separated jobs. The bigger the specialization is, the more repetitive the tasks are in the organization.

The division into departments groups different jobs that may coordinate their common tasks. That grouping can be done in different ways:

- *By functions,* where all the specialists are grouped in the same departments.
- *By product,* grouping tasks in departments by the product or service generated by the organization, increasing the responsibility by the achievement of the service.
- *By geography,* organising the departments by regions or territories.
- *By processes;* every department is specialized in one of the production phases.
- *By the type of client,* better satisfying the problems and needs of the clients.

The centralization is also an important element in the structure of the organizations, indicating where the decisions are taken. Centralized organizations take the decisions in only one place. In decentralised organizations the decisions are delegated to managers, located closer to the action.

The analysis of the structure determines the way the members of the organization are grouped, where the decisions are taken, and the relationships between the members of the organization.

In organizations of agents, the structure of the organization is normally defined in terms of roles and groups. Roles represent the

functionalities or activities of the agents. Groups specify the context for the activities of the agents. The communication is carried on within the groups [Dignum and Dignum, 2007a]. Thus, different dependencies are normally specified among the roles: heritage, compatibility, communication and coordination, authority, control, etc. These dependencies determine the relationships between the roles, which coordinate the actions of the agents.

The modelling language *MOISE-Inst* [Gateau *et al.*, 2005], offers one of the most complete specifications of the structure of an organization of agents. The structure of the MAS is defined by terms of roles, groups and relationships.

A role consists in a series of restrictions that an agent should follow to accept to be part of a group carrying that role. Those restrictions affect its relationships with other roles and its objectives and plans to follow.

A group is a set of relationships and roles, determining the cardinality restrictions (minimum and maximum number of agents playing a certain role in a group). The relationships of heritance and compatibility are also defined. Subgroups are also allowed.

Finally, social relationships determine the knowledge connections (what agents can obtain information from other agents), communication links (who is allowed to communicate with other agents) and authority relationships (who has control over others).

4.2.2. FUNCTIONALITY

In human organizations, the mission describes the reason for the existence of the organization, specifying the results (products or services) that proportionate. The groups of interest to whose it is dedicated and the global benefits expected to achieve are also specified. It determines the global objectives of the system; the services offered or required, as long as

the products associated to those services and the clients, users, etc. affected by the system.

Once the general purpose of an organization is known, it is possible to identify the basic functions needed to its achievement. The complexity of the design of the organization consists in reducing the general activity categories to specific subcategories. The final objective is to obtain individual tasks, that should be grouped to obtain the maximum productivity and efficiency with the minimum cost [Peiró, 1995].

In a similar way, in organizations of agents global objectives are also defined. Those objectives specify the general desired behaviour of the system. There are also particular objectives for roles and groups that establish a set of tasks and actions to achieve them.

In *MOISE-Inst* [Gateau *et al.*, 2005], the global objectives of the system are decomposed, through the use of plans, in specific objectives distributed among the agents. The plans describe the sequences of the objectives. Roles are assigned with a series of coherent objectives. The agent that plays that role must undertake to achieve those objectives.

Another important aspect in organizations is the concept of service. It is defined by a coherent block of functionality that is carried on by serving to other entity. Detailing the services offered by an organization will allow the agents of the system to discover, invoke, monitor or even compose them.

The specification of services has not been deeply considered by methodologies of agents, which are mainly centred in interaction protocols and in the tasks of roles and agents. Only AML [Cervenka and Trencansky, 2007], allows to specify what services are offered or required by the different entities of the system (roles, agents or organizing units). This Language uses its own models, based in UML.

4.2.3. COORDINATION

In human organizations, the coordination of tasks is obtained by three different mechanisms [Wagner, 2004]:

- *Mutual adaptation:* the members share the information related with their job and decide how to perform a tasks and who should perform it.
- *Direct supervision:* a person assumes the responsibility of the work of a group, acquiring the authority to decide what tasks must be done, who should perform them and how to relate the tasks to obtain the final result.
- *Normalization:* proportionate the standards and procedures to help the members of the organization to determine how to perform the tasks.

In organizations of agents, the coordination is generated by the use of social regulations. They must describe the expected behaviour of its members; the allowed, required and needed actions and those to be avoided. The sanctions to apply if not desirable actions are carried on should also be specified as well as the rewards to offer to the actions carried out by the procedure established in the regulations. Rules are normally defined and controlled by institutions with a legal status. Rules are essential to solve coordination problems in big and heterogeneous systems, where the social and direct control cannot be carried out [López *et al.*, 2006].

In *MOISE-Inst* [Gateau *et al.*, 2005], regulations define the permissions, obligations and prohibitions of the agents while playing a determined role or while being part of a group. Rules are related with the execution of certain objectives satisfying their mission, within a particular context and during an established period of time. The performance of the

rules is supervised by a specific role that may sanction a role affected by a rule.

In *Electronic Institutions* [Esteva *et al.*, 2001], there is a social layer formed by internal agents that know the interaction rules and grant that the interactions will be carried out according to those rules.

OperA [Dignum, 2004] proposes the establishment of interaction contracts to control the behaviour of the agents when they interact with each other. Those contracts describe the conditions and rules to apply while that interaction is produced.

4.2.4. SYSTEM DYNAMICS

In human organizations, every organization must allow its member to enter and leave the organization in a dynamic way. The organization incorporates members depending on their abilities, knowledge or aptitudes to obtain their purposes [Peiró, 1995].

In organizations of agents, control mechanisms should be established. Those mechanisms should control when the agents can enter the organization and their position within the organization (their roles and the groups in which they will enter). Expulsion processes must also be considered, when an agent carries out some anomalous behaviour within the organization. The dynamic aspect of the organization also implies the process of creation and elimination of the groups and units contained in the organization.

In *Electronic Institutions* [Esteva *et al.*, 2001], the agent *institution manager* controls the arrival of external agents. It creates an internal representative agent, called *governor*, for every external agent authorized to participate in the institution.

In *OperA* [Dignum, 2004], the agents are associated to the roles by establishing social contracts. Every contract describes the conditions and

rules that acquires an agent to play a role.

4.2.5. ENVIRONMENT

In human organizations, the environment covers all the elements outside the organization: suppliers, clients, rivals, government organisms, financial institutions and investors and the job market that provides the employees. Economic, geographical and political conditions are also part of the environment [Wagner, 2004].

The environment is, then, the source of needed resources to survive [Hodge *et al.*, 1998], providing the materials, technology and the members required to develop the products and services, as long as the enough number of clients to consume those products offering benefits to the organization.

In multiagent systems, the environment is mainly associated with the resources and applications that use the agents. In *Gaia* [Wooldridge *et al.*, 2000], the access modes to the resources are established (to read, interact, extract information, etc.).

AML [Cervenka and Trencansky, 2007] considers the sensors and actuators of the agents with their environment. Sensors should model the ability of the agents to observe, perceive states or receive signals; while actuators model their ability to produce certain effect over other objects or entities.

OMNI [Vázquez-Salceda *et al.*, 2005] establishes who are the *stakeholders* or groups of interest; those entities with certain requirements or needs over the system. The objectives and dependencies about the organization are also identified.

4.3. SUMMARY AND CONCLUSIONS

This fourth chapter develops the characteristics of the organizations of agents. First, the main features of the organizations are described, starting from the concept of organization, related with the human organizations, which are the origin of the organization of agents. Then the main factors of the organizations are developed, paying special attention to the following: s*tructure*, *functionality*, *normalization*, *dynamicity*, and *environment.*

As explained in this chapter, human organizations establish a series of mechanisms to restrict and control the activities to perform in order to coordinate them.

First, the specification of the objectives of the organization determines the tasks to be carried out. Those tasks require certain roles well differentiated, each of those has one or more activities assigned for specific situations. Those roles generate a structure that allows the coordination of the activities and the transmission of information.

In second place, the organization has selection systems to incorporate new members. Those whose conducts are more appropriated are chosen.

Finally, the organization has training and socialization mechanisms, not only related with the tasks, but also about roles, rules and values, to create a group environment. Groups are composed by a limited number of individuals with common interactions and certain degree of shared rules.

Organizations are a useful paradigm to analyze and design MAS [Van Den Broek *et al.*, 2006], limiting the range of the interactions of the agents and providing interaction patrons previously established. Organizations also offer mechanisms to divide the tasks and to generate a more specialized work. Thus they allow formalizing the objectives of the system at a high level, establishing the purpose of the organization. Units or groups contained in the organization, generate certain visibility limits, allowing the internal

agents of every unit to know its internal structure, but it is not visible to external agents [Ferber *et al.*, 2004].

V. Dignum affirms that the organizations of agents represent a step forward for multiagent systems, allowing the coordination and collaboration of open systems [Dignum and Dignum, 2007a]. The organization exists independently of the agents that participate in it. Those agents will enter or leave the organization in a dynamic way. Thus, it is assumed the existence of global objectives that determine the existence of the organization. As an open system, it allows the arrival of new agents that will require a registration by contracts, specifying their interests and abilities.

Organizations represent one crucial aspect in the architecture presented in this document. The fact that the agents can work together and can share objectives and a way of responding to requests it is important in this architecture. Agents are simpler elements that could have solved the same problem but in a more complicated and risky way. Using organizations allow the architecture to simplify the interaction between the agents by grouping them. As it will be explained in the next section, the main organizations that form the OBaMADE architecture can communicate among them by using communication mechanisms that could not have been possible (or, at least, it would have been much harder to achieve) by using only individual agents.

Now, after explaining the main characteristics of the distributed environments, and the specific features both of the multiagent systems and of the organizations of agents, the OBaMADE architecture will be explained in the next chapter. That architecture will be applied to dynamic distributed environments, where different people are involved at the same time with different roles, and with different kind of interaction with the system.

”Our problems are man-made, therefore they may be solved by man. And a man can be as big as he wants. No problem of human destiny is beyond human beings.” *John F. Kennedy*

5. The OBaMADE Architecture

In this chapter the OBaMADE (Organization Based MultiAgent Architecture for Distributed Environments) architecture is fully described. First the main structure of the architecture is developed. Then, the different elements are explained, including all the components, from external interface agents to the inner service oriented structure, where all the requests are solved.

OBaMADE (Organization Based Multiagent Architecture for Distributed Environments) represents a new architecture to face problems that involve a great variety of people, data that can originate from different sources, and solutions that may be requested from different locations at the same time.

The OBaMADE architecture exposed in this thesis uses the distributed capabilities of an organization of agents combined with the generalization and knowledge extraction power of the Case-Based Reasoning methodology. Thus, the architecture is divided in distinctly different parts, where the

external agents are in charge of the communications with the external sources of information or requests. The internal elements of the architecture represent the communication components and the services that treat the information and the request, following the CBR paradigm.

The OBaMADE architecture makes use of the techniques explained in previous chapters. Its aim is to solve problem in distributed environments, where information may change in real time and where there are different sources of information and of requests to the system. The main elements of the architecture explained in this chapter are:

- *The Interface Agent Organization:* a set of agents that recover the information that may be entered into the system. That information can be either an input of new data, a request of a service or an answer from a request done by the system.
- *The CBR-Services Organization:* a set of services coordinated by communication and control agents. This organization uses an internal CBR methodology to extract all the possible knowledge from the available data.
- *The Additional Services Organization:* the services covering the CBR basic methodology may involve some other services that may be needed by the systems developed with this architecture. These specific services, which can be modelled for any application, are coordinated and communicated by agents that share part of the information with the CBR services.
- *The Communication Organization:* serves as an interconnection between the other elements of the architecture, helping to interchange the information and solving the needs of services from the agents.

The current chapter begins with the description of the main structure of the architecture, where the different components will be shown, and the basic interaction between them will be explained. Next, the agents involved in the organization will be fully described, giving details of the way they work in their different tasks. Afterwards, the services that comprise the core of the system will be explained, with special attention to the way the information is treated in order to obtain proper solutions to the proposed problems. Finally, the applications of the OBaMADE architecture are detailed, explaining how it can be adapted to solve different kind of problems regarding information treatment.

5.1. ARCHITECTURE DESCRIPTION

The OBaMADE architecture was primarily designed to develop Distributed System applications. These applications must be dynamic, flexible, robust, adaptable to changes in context, scalable and easy to use and maintain. However, the architecture can be used to develop any kind of complex systems because it is capable of integrating almost any service and application desired, with no dependency on any specific programming language. Because the architecture acts as an interpreter, the users can run applications and services programmed in virtually any language, but have to follow a communication protocol that all applications and services must incorporate.

Another important functionality is that, because of the agents' capabilities, the systems developed can make use of reasoning mechanisms or learning techniques to handle services and applications according to context characteristics, which can change dynamically over time. Agents, applications and services can communicate in a distributed way, even from mobile devices. This makes it possible to use resources no matter their location. It also allows the starting or stopping of agents, applications,

services or devices separately, without affecting the rest of resources, so the system has an elevated adaptability and capacity for error recovery.

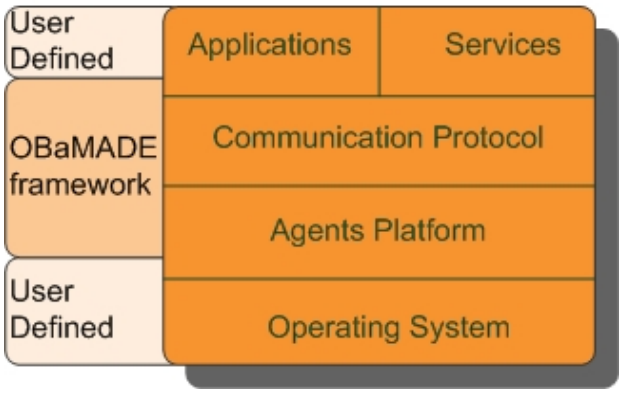


Figure 13. *OBaMADE framework.*

As can be seen on *figure 13*, the OBaMADE framework defines four basic blocks: Applications, Services, Agents Platform and Communication Protocol. These blocks provide all the functionalities of the architecture:

- *Applications.* These represent all the programs that can be used to exploit the system functionalities. Applications are dynamic and adaptable to context, reacting differently according to the particular situations and the services invoked. They can be executed locally or remotely, even on mobile devices with limited processing capabilities, because computing tasks are largely delegated to the agents and services.
- *Agents Platform.* This is the core of OBaMADE, integrating a set of agents, each one with special characteristics and behaviour. An important feature in this architecture is that the agents act as controllers and administrators for all applications and services, managing the adequate functioning of the system, from services, applications, communication and performance to reasoning and decision-making. In OBaMADE, services are managed and

coordinated by deliberative BDI agents. The agents modify their behaviour according to the users' preferences, the knowledge acquired from previous interactions, as well as the choices available to respond to a given situation.

- *Services.* These represent the activities that the architecture offers. They are the bulk of the functionalities of the system at the processing, delivery and information acquisition levels. Services are designed to be invoked locally or remotely. Services can be organized as local services, web services, GRID services, or even as individual stand alone services. Services can make use of other services to provide the functionalities that users require. OBaMADE has a flexible and scalable directory of services, so they can be invoked, modified, added, or eliminated dynamically and on demand. It is imperative that all services follow the communication protocol to interact with the rest of the architecture components.
- *Communication Protocol.* This allows applications and services to communicate directly with the agents platform. The protocol is completely open and independent of any programming language. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications [Cerami, 2002]. Services and applications communicate with the agents platform via SOAP messages. A response is sent back to the specific service or application that made the request. All external communications follow the same protocol, while the communication among agents in the platform follows the FIPA Agent Communication Language (ACL) specification. This is especially useful when applications run on limited processing capable devices (e.g. cell phones or PDAs). Applications can make

use of agents platforms to communicate directly (using FIPA ACL specification) with the agents in OBaMADE, so while the communication protocol is not needed in all instances, it is absolutely required for all services.

Users can access the system through distributed applications, which run on different types of devices and interfaces (e.g. computers, cell phones, PDA). *Figure 14* shows the basic schema of OBaMADE where all requests and responses are handled by the agents in the platform. The agents analyze all requests and invoke the specified services either locally or remotely. Services process the requests and execute the specified tasks. Then, services send back a response with the result of the specific task.

OBaMADE is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents [Bratman *et al.*, 1988, Pokahr *et al.*, 2003]. Deliberative BDI agents are able to cooperate, propose solutions on very dynamic environments, and face real problems, even when they have a limited description of the problem and few resources available.

These agents depend on beliefs, desires, intentions and plan representations to solve problems [Bratman, 1987, Georgeff and Rao, 1998]. Deliberative BDI agents are the core of OBaMADE. There are different kinds of agents in the architecture, each one with specific roles, capabilities and characteristics. This fact facilitates the flexibility of the architecture in incorporating new agents.

The agents that form part of the agents' platform interact with each other to coordinate the requests received and to communicate between the interface agents and the services provided by the architecture. The location of those agents in the agents' platform can be seen in *figure 14.*

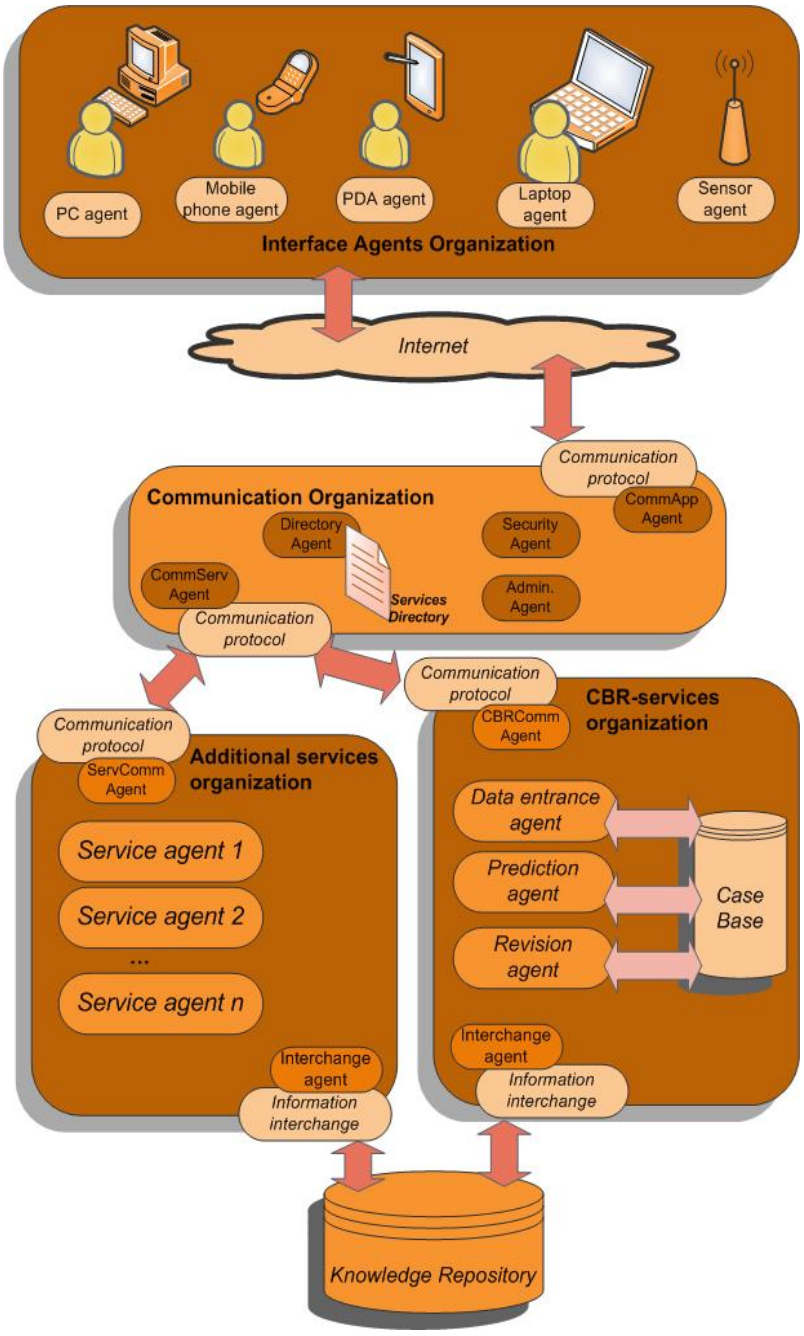


Figure 14. *OBaMADE basic schema.*

The information flow is started by the users, which introduce in the system, through the *Interface Organization*, their requests. Once the request is processed by the *Interface Organization*, it is send to the *Communication Organization*, that decides which service is in charge of the tasks required by the user. Then, the request is sent to one of the *Services Organizations*, depending on the request generated by the user.

When the request is accomplished, it is returned back to the user through both the *Communication Organization* and the *Interface Organization.* A basic schema of this information flow is shown in *figure 15.* The elements and transfers in *figure 15* will be deeply explained in next sub-sections.

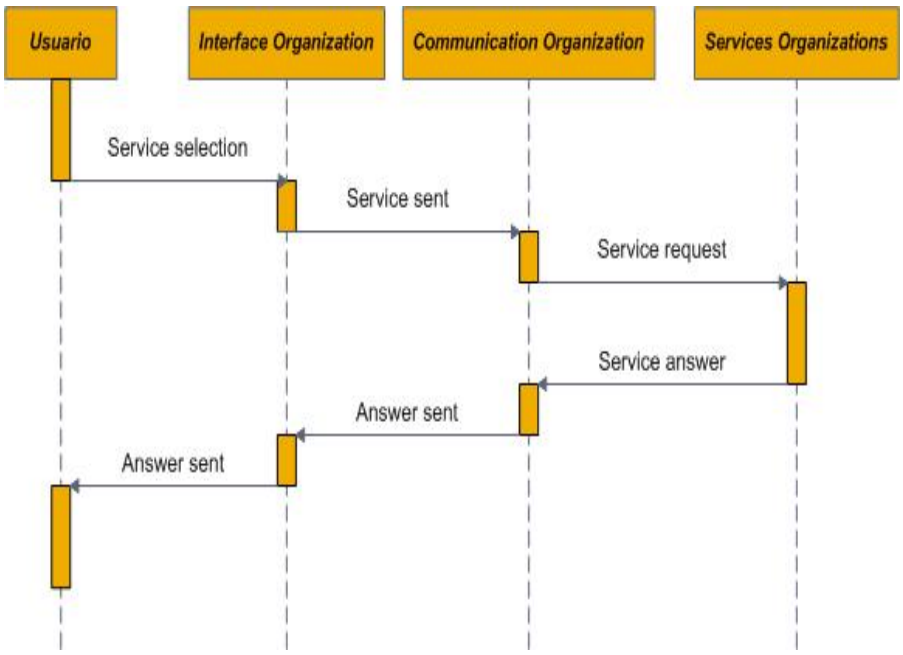


Figure 15. *OBaMADE basic information flow.*

5.2. INTERFACE AGENTS ORGANIZATION

Interface agents were designed to be embebbed in user applications. Interface agents communicate directly with the agents in the communication organization, so there is no need to employ the communication protocol, the FIPA ACL specification is used indeed.

The requests are sent directly to the Security Agent, which analyzes the requests and sends them to the Manager Agent. The rest of the process follows the same guidelines for calling any service. These agents must be simple enough to allow them to be executed on mobile devices, such as cell phones or PDAs.

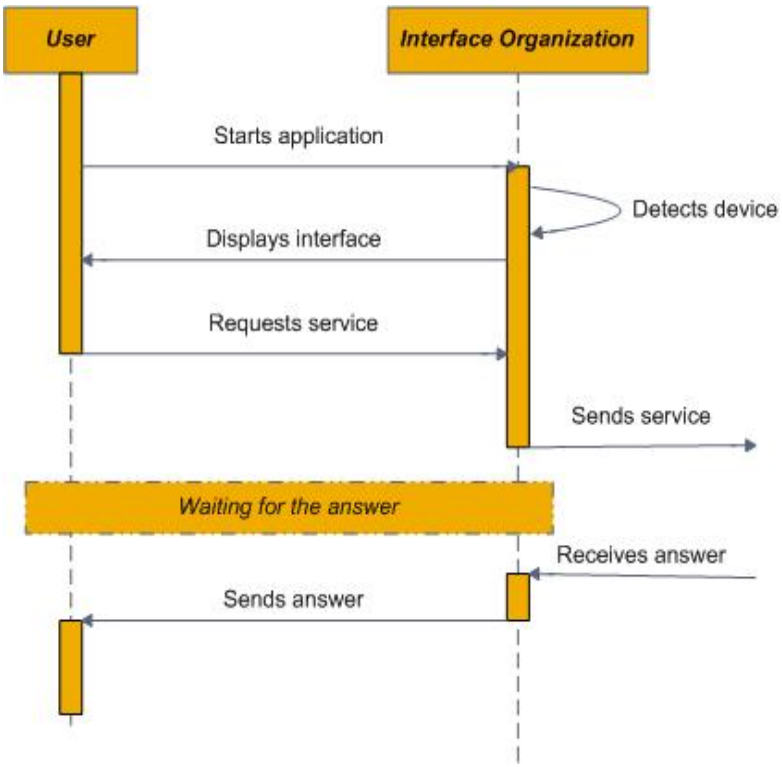


Figure 16. *Interface Organization activity.*

The *Interface Organization* receives information from the users. When a users starts an application, it should detect the kind of device that is requesting a service to properly sent it the interface according to the used device. Then, the users introduces the kind of requests that is demanding. The *Interface Organization* receives the request and sends it to the *Communication Organization,* that will solve it by using different services of the available *Services Organizations.* When the *Communication Organization* sends request answer to the *Interface Organization* that sends it finally to the user. This sequence of transfers can be seen in *figure 16.*

OBaMADE is an open architecture that allows developers to modify the structure of these agents, so that agents are not defined in a static manner. Developers can add new agent types or extend the existing ones to conform to their project needs. However, most of the agents' functionalities should be modelled as services, releasing them from tasks that could be performed by services. Services represent all functionalities that the architecture offers to users and uses itself. As previously mentioned, services can be invoked locally or remotely. All information related to services is stored into a directory which the platform uses in order to invoke them, i.e., the services. This directory is flexible and adaptable, so services can be modified, added or eliminated dynamically. Services are always on "listening mode" to receive any request from the platform. It is necessary to establish a permanent connection with the platform using sockets.

Every service must have a permanent listening port open in order to receive requests from the platform. Services are requested by users through applications, but all requests are managed by the platform, not directly by applications. When the platform requests a service, the *CommServ* Agent sends an XML message to the specific service. The message is received by the service and creates a new thread to perform the task. The new thread has an associated socket which maintains open communication with the platform

until the task is finished and the result is sent back to the platform. This method provides services capable of managing multiple and simultaneous tasks, so services must be programmed to allow multi-threading.

However, there could be situations where multi-tasks are not being permitted, for instance high demanding processes where multiple executions could significantly reduce the services performance. In these cases, the *Manager Agent* asks the *CommServ Agent* to consult the status of the service, which informs the platform that it is busy and cannot accept other requests until finished. The platform must then seek another service that can handle the request, or wait for the service to be idle. To add a new service, it is necessary to manually store its information into the directory list managed by the Directory Agent. Then, CommServ Agent sends a ping message to the service. The service responds to the ping message and the service is added to the platform. A service can be virtually any program that performs a specific task and shares its resources with the platform. These programs can provide methods to access data bases, manage connections, analyze data, get information from external devices (e.g. sensors, readers, screens, etc.), publish information, or even make use of other services. Developers are free to use any programming language. The only requirement is that they must follow the communication protocol based on transactions of XML (SOAP) messages.

5.3. COMMUNICATION ORGANIZATION

In the middle of the OBaMADE structure there is an organization designed to establish correct communications between the rest of the elements of the architecture. *Figure 17* shows a schema of how the agents that form this organization may be structured within the organization. The interchange of information from the interface organization to the organizations in charge of the service passes through this organization, where

specific agents must make certain decisions, as will be explained next.

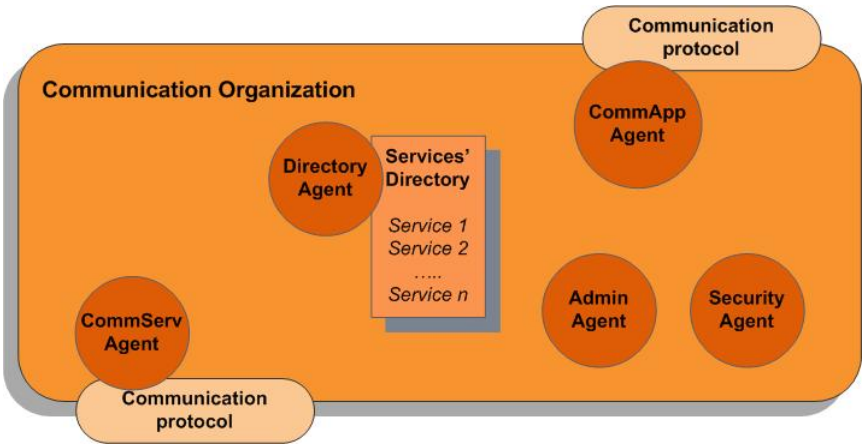


Figure 17. *Communication Organization schema.*

The agents that form this organization have the following descriptions and tasks to perform:

- *CommApp Agent.* This agent is responsible for all communications between applications and the platform. It manages the incoming requests from the applications to be processed by services. It also manages responses from services (via the platform) to applications. *CommApp Agent* is always on *"listening mode"*. Applications send XML messages to the agent requesting a service, after which the agent creates a new thread to start communication by using sockets. The agent sends all requests to the Manager Agent, which processes the request. The socket remains open until a response to the specific request is sent back to the application using another XML message. All messages are sent to *Security Agent* for their structure and syntax to be analyzed.
- *CommServ Agent.* It is responsible for all communications between services and the platform. The functionalities are similar to *CommApp Agent* but backwards. This agent is always on *"listening*

mode" waiting for responses of services. *Manager Agent* signals to *CommServ Agent* which service must be invoked. Then, *CommServ Agent* creates a new thread with its respective socket and sends an XML message to the service. The socket remains open until the service sends back a response. All messages are sent to *Security Agent* for their structure and syntax to be analyzed. This agent also periodically checks the status of all services to know if they are idle, busy, or crashed.

– *Directory Agent.* It manages the list of services that can be used by the system. For security reasons [Snidaro and Foresti, 2007], the list of services is static and can only be modified manually; however, services can be added, erased or modified dynamically. The list contains the information of all trusted available services. The name and description of the service, parameters required, and the IP address of the computer where the service is running are some of the information stored in the list of services. However, there is dynamic information that is constantly being modified: the service performance (average time to respond to requests), the number of executions, and the quality of the service. This last data is very important, as it assigns a value between 0 and 1 to all services. All new services have a quality of service (QoS) value set to 1. This value decreases when the service fails (e.g. service crashes, no service found, etc.) or has a subpar performance compared to similar past executions. QoS is increased each time the service efficiently processes the tasks assigned. Information management is especially important in distributed environments because the data processed is very sensitive and personal. Thus, security must be a major concern when developing systems related with distributed environments. For this reason OBaMADE does not

implement a service discovery mechanism requiring systems to employ only the specified services from a trusted list of services. However, agents can select the most appropriate service (or group of services) to accomplish a specific a task.

- *Supervisor Agent.* This agent supervises the correct functioning of the other agents in the system. *Supervisor Agent* periodically verifies the status of all agents registered in the architecture by sending ping messages. If there is no response, the *Supervisor Agent* kills the agent and creates another instance of that agent.
- *Security Agent.* This agent analyzes the structure and syntax of all incoming and outgoing XML messages. If a message is not correct, the *Security Agent* informs the corresponding agent (CommApp or CommServ) that the message cannot be delivered. This agent also directs the problem to the *Directory Agent,* which modifies the QoS of the service where the message was sent.
- *Manager Agent.* Decides which agent must be called by taking into account the QoS and user preferences. Users can explicitly invoke a service, or can let the *Manager Agent* decide which service is best to accomplish the requested task. If there are several services that can resolve the task requested by an application, the agent selects the optimal choice. An optimal choice has higher QoS and better performance. *Manager Agent* has a routing list to manage messages from all applications and services. This agent also checks if services are working properly. It requests the *CommServ Agent* to send ping messages to each service on a regular basis. If a service does not respond, *CommServ* informs *Manager Agent*, which tries to find an alternate service, and informs the *Directory Agent* to modify the respective QoS.

The *Communication Organization* receives the user's request from the *Interface Organization*. When the request arrives at the *Communication Organization* it should send it to the appropriate service. That service can be on in the *CBR Services Organization* or in the *Additional Services Organization.* The *Communication Organization* should coordinate the dataflow from the exterior of the system and to the internal services. This dataflow can be seen in *figure 18.*

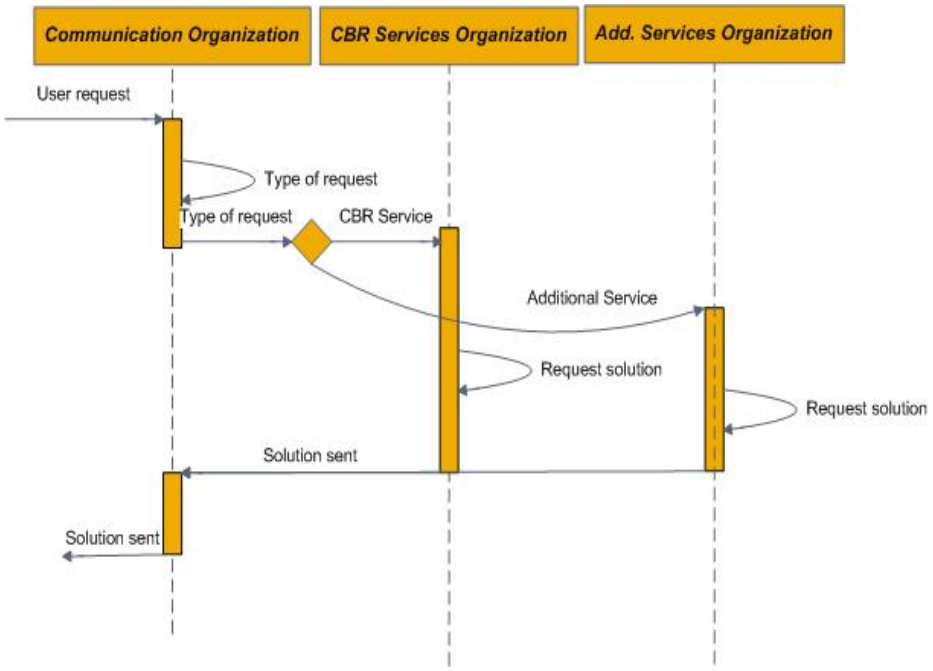


Figure 18. *Communication Organization dataflow.*

5.4. CBR SERVICES ORGANIZATION

The reasoning capabilities of the OBaMADE architecture are based on the Case-Based Reasoning methodology. The main basic aspects of this methodology are explained in *Appendix C*. The CBR methodology uses past information to solve new problems. The use of past information combined

with an appropriate set of artificial intelligence techniques produces a successful knowledge extraction. It is essential to transform the information, i.e. the data, into knowledge. When data can be used to solve problems, then it is more than data. This transformation can be properly executed with a methodology like CBR.

The four main phases of the basic CBR cycle should be taken into account in order to accomplish the CBR methodology. In this case, the phases are transformed into *services* that respond to requests made by the *interface agents*, being redirected by the *communication organization*. The data flow from the *communication organization* into the CBR services is shown in *figure 19.* There can be seen the input of the request from the *communication organization* and how it is treated by the different services of the *CBR services organization.*

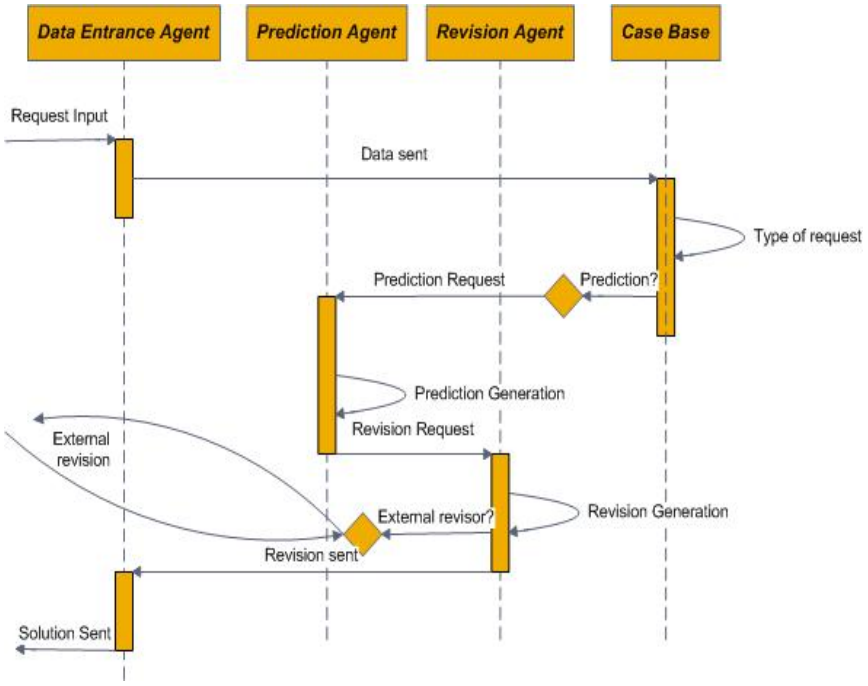


Figure 19. *CBR Services Organization dataflow.*

Next, the adaptation of the CBR phases to the OBaMADE architecture is explained, focusing on the artificial intelligence techniques employed to obtain the best results from the available information. First, the organization and creation of the case base are explained, paying special attention to the structure of the case base and the advantages of properly organizing the stored data. Then the introduction of information is analyzed, specifying the process carried out to enrich the case base. The third phase described is the generation of a solution from a request arrived at the system; the main steps taken by the request until the arrival of the final solution are described. Finally, the revision process, where the proposed solution is validated, is described.

5.4.1. ORGANIZING THE CASE BASE

Case-Based Reasoning is a methodology that depends on past stored data from which knowledge is extracted in order to solve new problems. It is thus critical to properly organize the case base, the structure where the information is kept [Sun *et al.*, 2004]. Here, a new extension on the well-known Self-Organizing Map algorithm is presented [Kohonen, 1995]. The algorithm has a double purpose: first, it is used to sort out all the information that is stored in the case base. Then, it is used to retrieve the most similar cases to the problem introduced in the system that needs to be solved.

The SOM is based on a type of unsupervised learning called competitive learning; an adaptive process in which the neurons in a neural network gradually become sensitive to different input categories, sets of samples in a specific domain of the input space. The main feature of the SOM algorithm is that the neighbours on the lattice, as well as the winning neuron, are also allowed to learn – i.e. to adapt their characteristics to the input. Thus, the neighbouring neurons gradually come to represent similar

inputs, and their representations become ordered on the map lattice.

The difference between the SOM and the WeVOS hence lies in the update of the weights of the neighbours of the winner neuron as can be seen from *Eqs. (1)* and *(2).*

Update of neighbourhood neurons in SOM:

$$w_k(t+1) = w_k(t) + \alpha(t)\eta(v,k,t)\big(x(t) - w_v(t)\big) \quad (1)$$

Update of neighbourhood neurons in WeVOS:

$$w_k(t+1) = w_k(t) + \alpha(t)\eta(v,k,t)\left(\big[x(t) - w_v(t)\big] + \big[w_v(t) - w_k(t)\big]\left(\frac{d_{vk}}{\Delta_{vk}\lambda} - 1\right)\right) \quad (2)$$

where w_v is the winning neuron, α the learning rate of the algorithm, $\eta(v,k,t)$ is the neighbourhood function (usually, the Gaussian function or a difference of Gaussians), where v represents the position of the winning neuron in the lattice and k the positions of the neurons in the neighbourhood of this one, x is the input to the network and λ is a *"resolution"* parameter, d_{vk} and Δ_{vk} are the distances between the neurons in the data space and in the map space respectively.

The idea behind the WeVoS meta-algorithm is to apply the scheme of an ensemble of classifiers working together to solve a single classification problem [Heskes, 1997, Ayd *et al.*, 2009] to the topology preserving algorithms. An ensemble of maps can be trained on a dataset, and a final map summarizing the main features detected by each one can be calculated.

The WeVoS fusion algorithm presented in this study aims to obtain the final map by using the information contained in the maps composing the ensemble on a unit-by-unit basis. Usually, the final characteristics vectors of a single map are calculated from a single training over the

dataset.

The WeVoS algorithm tries to generate the final characteristics vector for each unit by relying on an informed decision about the adaptation of its homologous units from an ensemble of maps, each of which has been trained on slightly different parts of the dataset [Breiman, 1996]. This vector is also recalculated for the neighbours of the unit.

As a result, the final map obtained not only determines the best position for each unit based on an informed decision, but also maintains one of the most important features of this type of algorithms: its topological ordering. WeVoS is an improved version of the superposition algorithm presented in several previous works [Baruque *et al.*, 2007]. Although it has been successfully applied to the analysis of real-life data [Baruque *et al.*, 2008], in this study it is applied for the first time to solve this kind of practical problem.

The first step in this meta-algorithm is to calculate the *"quality"* of each of the units comprising each map, in order to rely on some kind of informed decision for the fusion of units. This *"quality"* measure (or error measure) could be any one of the many *"quality of map"* measures presented in scientific literature regarding Self-Organizing Maps [Polani, 2001, Polzlbauer, 2004]; provided that it may be calculated on a unit-by-unit basis.

The final map is obtained again on a unit-by-unit basis. Firstly, the units of the final map are initialized by calculating the centroids of the units in the same position of the map grid in each of the trained maps. Then, the final position of that unit is recalculated using the information associated with the units in that same position in each of the ensemble maps. For each unit, a voting process is performed as shown in *Eq. 3*:

$$V_{p,m} = \frac{\sum b_{p,m}}{\sum_{i=1}^{M} b_{p,i}} \cdot \frac{q_{p,m}}{\sum_{i=1}^{M} q_{p,i}} \quad (3)$$

where, $V_{p,m}$ is the weight of the vote for the unit included in map m of the ensemble, in its position p; M is the total number of maps in the ensemble; $b_{p,m}$ is the binary vector used for marking the dataset entries recognized by the unit in position p of map m; and, $q_{p,m}$ is the value of the desired quality measure for the unit in position p of map m.

Algorithm 1. *Weighted Voting Superposition (WeVoS).*

1: train several networks by using the bagging (re-sampling with replacement) meta-algorithm

2: **for each map** (*m*) **in the ensemble**

    3: **for each unit position** (*p*) **of the map**

    4: calculate the quality measure/error chosen for the current unit

    5: **end**

6: **end**

7: calculate an accumulated total of the quality/error for each position *Q(p)* on all maps

8: calculate an accumulated total of the number of data entries recognized by a position on all maps *D(p)*

9: **for each unit position** (*p*)

    10: initialize the fused map (*fus*) by calculating the centroid (*w'*) of the units of all maps in that position (*p*)

11: **end**

12: **for each map** (*m*) **in the ensemble**

13: **for each unit position** (*p*) **of the map**

14: calculate the vote weight of the (*p*) in the map (*m*) by using Eq. 2

15: feed the weights vector of the (*p*) to the fused map (*fus*), as if it were a network input, using the weight of the vote calculated in step 14 as the learning rate and the index of that same (*p*) as the index of the BMU.

16: **end**

17: **end**

The weights of each unit are fed into the final network in the same way as the data inputs during the training phase of a SOM, considering the *'homologous'* unit in the final map as the Best Matching Unit (BMU). The weights of the final unit will be updated towards the weights of the composing unit. The difference in the updating performed for each homologous unit that forms part of the map depends on the quality measure calculated for each unit: the higher the quality (or the lower the error) of the unit in the composing map, the stronger the updating of the unit in the summary map towards the weights of that particular unit. With respect to quality determination, a single quality measure or a linear combination of several measures may be used. The number of data inputs recognized by each unit is also taken into account in the quantization of the 'most suitable' unit among those competing for the same position in the final map. In short, the summarization algorithm considers the most suitable weights of a composing unit to be the weights of the unit in the final map, according to both the number of inputs recognized and the adaptation quality of the unit. The model, referred to as WeVoS, is described in detail in the *algorithm 1*.

This new approach not only takes the characteristics of each unit into account, but also the topographic ordering of its neighbourhood. The approach is intended to generate more meaningful maps by representing the inner structure of the dataset more faithfully. Those capabilities are a great added value to a CBR system since they facilitate the creation of the structure of the case base, where grouping similar cases together is a great advantage. They are also important when trying to recover the most similar cases to the problem introduced in the system, because of the increased speed of the recovery that results when similar cases are close one to another.

5.4.2. DATA ENTRANCE AGENT

Case-Based Reasoning systems are highly dependent on stored information. The novel algorithm presented here, Weighted Voting Summarization of SOM ensembles (WeVoS-SOM) [Baruque *et al.*, 2009] is used to organize the data that is accumulated in the case base. It is also used to recover the most similar cases to the proposed problem.

The main objective of the WeVoS-SOM is to generate a final map processing several other similar maps unit by unit. Instead of trying to obtain the best position for the units of a single map trained over a single dataset, it aims to generate several maps over different parts of the dataset. Then, it obtains a final summarized map by calculating by consensus which is the best set of characteristics vector for each unit position in the map. To perform this calculation, this meta-algorithm must first obtain the *"quality"* [Polzlbauer, 2004] of every unit that composes each map, so that it can relay in some kind of informed resolution for the fusion of neurons.

The final map obtained is generated unit by unit. The units of the final map are first initialized by determining their centroids in the same position of the map grid in each of the trained maps. Afterwards, the final

position of that unit is recalculated using data related to the unit in that same position in each of the maps of the ensemble. For each unit, a sort of voting process is carried out as shown in *Eq. 3.*

The final map is fed with the weights of the units, as it is done with data inputs during the training phase of a SOM [Kohonen, 1995], considering the "homologous" unit in the final map as the BMU. The weights of the final unit will be updated towards the weights of the composing unit. The difference of the updating performed for each *"homologous"* unit in the composing maps depends on the quality measure calculated for each unit. The higher the quality (or the lowest error) of the unit of the composing map, the stronger the unit of the summary map will be updated towards the weights of that unit. The summarization algorithm will consider the weights of the "*most suitable"* composing unit to be the weights of the unit in the final map according to both the number of inputs recognized and the quality of adaptation of the unit (*Eq. 3*). The expected result of this new approach is to obtain maps that are more true to the inner structure of the dataset.

5.4.3. SOLUTION REQUEST AGENT

When a prediction is requested by a user, the system begins by searching the case base to recover the most similar cases to the problem proposed. Then, it creates a prediction using artificial neural networks. Once the most similar cases are recovered from the case base, they are used to generate the solution. Growing RBF networks [Ros *et al.*, 2007] are used to obtain the predicted future values corresponding to the proposed problem.

This adaptation of the RBF networks allows the system to grow during training, gradually increasing the number of elements (prototypes) which play the role of the centres of the radial basis functions. The creation

of the Growing RBF must be made automatically, which implies an adaptation of the original GRBF system. The error for every pattern is defined by (Eq. 3).

$$e = 1/n \sum_{k=1}^{p} \| t_{ik} - y_{ik} \| \tag{3}$$

where t_{ik} is the desired value of the k_{th} output unit of the i_{th} training pattern, y_{ik} the actual values of the k_{th} output unit of the i_{th} training pattern.

The Growing RBF pseudocode is as follows in *Algorithm 2:*

Algorithm 2 . *Growing Radial Basis Function pseudocode.*

1: Calculate the error, e_i (Eq. 3) for every new possible prototype. a. If the new candidate is not among those selected and the error calculated is less than a threshold error, then the new candidate is added to the set of accepted prototypes. b. If the new candidate already belongs to the accepted ones and the error is less than the threshold error, then modify the weights of the units in order to adapt them to the new situation.
2: Select the best prototypes from the candidates ✓ If there are valid candidates, create a new cell centred on the valid candidate. ✓ Else, increase the iteration factor. If the iteration factor reaches 10% of the training population, freeze the process.
3: Calculate global error and update the weights. ✓ If the results are satisfactory, end the process. If not, go back to step 1.

Once the GRBF network is created, it is used to generate the solution to the proposed problem. The solution proposed is the output of the GRBF network created with the retrieved cases. The GRBF network receives the values stored in the case base as input. With those values, the

network generates the proposed solution, using only the data recovered from the case base in previous phases.

5.4.4. REVISION AGENT

After generating a prediction, the system needs to validate its correction. OBaMADE can also query an expert user to confirm the automatic revision previously done. The system also provides an automatic method of revision that must be checked as well by an expert user which confirms the automatic revision.

Explanations are a recent revision methodology used to check the correction of the solutions proposed by CBR systems [Plaza *et al.*, 2005]. Explanations are a kind of justification of the solution generated by the system. To obtain a justification to the given solution, the cases selected from the case base are used again. As explained before, a relationship between a case and its future situation can be established.

If both the situations defined by a case and the future situation of that case are considered as two vectors, a distance between them can be defined, calculating the evolution of the situation in the considered conditions. That distance is calculated for all the cases retrieved from the case base that are similar to the problem to be solved. If the distance between the proposed problem and the solution given is not greater than the average distances obtained from the selected cases, then the solution is a good one, according to the structure of the case base.

If the proposed prediction is accepted, it is considered to be a good solution to the problem and can be stored in the case base in order to solve new problems. It will have the same category as the historical data previously stored in the system.

Algorithm 3. *Explanations pseudocode.*

1: For every selected case in the retrieval phase, the distance between the case and its solution is calculated.
2: The distance between the proposed problem and the proposed solution is also calculated.
3: If the difference between the distance of the proposed solution and that of the selected cases is below a certain threshold value, then the solution is considered to be valid.
4: If not, the user is informed and the process goes back to the retrieval phase, where new cases are selected from the case base.
5: If after a series of iterations the system does not produce a good enough solution, then the user is asked to consider accepting the best of the generated solutions.

5.5. ADDITIONAL SERVICES ORGANIZATION

The CBR services organization includes all the services related to the CBR methodology, with the four phases of the CBR cycle. The *case base* is only consulted by the services contained in that organization.

But there are more possible services that may use some other kind of information. The *knowledge repository* stores all the information treated by the architecture, including not only the cases, but also all the requests performed, and the consults made by the experts. It is a kind of big repository, containing an updated version of the case base and a complete log of all the activities carried out by the architecture.

So, depending on the specific application of the architecture, this organization may contain different services, most of them regarding the *knowledge repository* information.

Some of the possible services contained in this organization are:

- *Specific log reports:* every user of the system can consult its interactions with the system. The administrators can consult all the information stored in the knowledge repository. It can be used to create activity reports or to check the correct working the systems.
- *Information retrieval:* some historical information about a specific problem can be retrieved from the knowledge repository without necessarily being a request for a solution. It can be employed to consult information about the problem, to create statistical reports or to check the information stored and compare it with present values...
- *Consult previous actuations:* experts, that are requested to validate the solutions automatically generate by the system, can consult their previous validations to confirm their impressions or even to confirm the way to proceed.
- *Consult previous solutions:* when requesting a solution to a problem, it can be applied to consult previous solutions given to the similar problems.

These and other services can be created and included in the additional services organization to adapt the OBaMADE architecture to the specific problems it can be applied to.

5.6. APPLICATIONS

The OBaMADE architecture integrates organizational capabilities that allow the systems created based on this architecture to structure their components (mainly agents and services). The different elements integrated

in the architecture ensure the capability of offering communication services to different users and an internal structure of information that may be adapted to different problems. The kinds of problems that this architecture can solve are normally related to distributed environments, where the information can be obtained from different sources at the same time. Knowledge extraction is also one of the main fields where this architecture may be applied. The internal CBR structure of the services and of the management of the information allows the system to apply the capabilities of the CBR methodology to different fields.

The main applications of this architecture are the following: *prediction generation, classification, clustering* and *planning.*

This application fields will be explained next, developing how the described architecture may be easily adapted in order to solve the different kind of problems proposed.

5.6.1. PREDICTION GENERATION

The application field was tested with the two case studies that will be explained in the next chapter. This application of OBaMADE has produced great results, which will be explained in the next chapter.

The next section explains the adaptation of the architecture to this application, focusing on the data stored in the case base, the entrance of information and the generation of a solution.

The case base stores information with temporal parameters, in order to create temporal relationships between one moment and the immediate subsequent moment. The case base simultaneously stores the information about the knowledge field that is treated, and parameters regarding the time (date, hour... depending on the problem to be solved). This is how the CBR system structures the proposed problem (present situation) and its solution (future situation). Given a proposed problem, the system searches the case

base for future situations associated with the introduced problem.

The information is introduced in the case base by different means. It is possible to acquire information directly from users, who will introduce the information through their devices. But it is also possible to obtain information from satellites with online information services, and from specific sensors that may measure some interesting parameters. All that information is structured into the case base, keeping the temporal relationship between the parameters stored.

The system generates a solution after a solution request is received. The data introduced in the system to obtain a solution is a present situation, with values for some of the stored parameters. Then the system tries to recover the rest of the information, if possible, from other sources, like sensors or satellites. Once the information is organized, then the system recovers from the case base those situations more similar to the introduced. The system then generates a solution, applying the artificial intelligence techniques previously explained, by treating the recovered cases.

5.6.2. CLASSIFICATION AND CLUSTERING

Classification consists of structuring the information into one of a certain number of possible classes depending on its characteristics. Clustering consists of determining the possible different groups of elements from a set of elements. Those two techniques are highly related, and the OBaMADE architecture may serve to combine them to generate complex data mining applications.

In the case base creation phase, the available data is structured into the case base. If the existing data can be organized into clusters, the internal structure of the case base will reflect it.

Case base creation: this is the phase when the system determines the cluster in which the data is divided. While the information is incorporated

into the case base, it is located depending on the values stored, after which the clusters appear. Visualization algorithms may be needed to check the existence of the clusters and to give visual evidence of their creation. A new parameter can be stored into the case base, identifying the cluster in order to simplify the retrieval and the categorization of the information.

The case base stores the introduced data according to its characteristics. Similar data will be located close one to another. This will help to identify the clusters and to perform the classification tasks.

When new data arrives to the system, it is categorized and located in a specific cluster, if possible. New data may create a new cluster or, in case of strange data that the system does not consider to be compatible with the stored information, it can be rejected (the user is previously consulted to validate the decision taken by the system). So when new information is stored in the system, classification is automatically performed.

If a new element needs to be classified, it is compared to the elements stored in the case base. Then, the most similar elements to the new one will be identified with a cluster. The new element will now belong to the same cluster as those similar cases stored in the case base. The new case will be stored in the case base, to be used in future problems as part of the solution.

5.6.3. PLANNING

Planning may be integrated in the OBaMADE architecture. It consists only of changing the internal methodology from case-based reasoning, to case-based planning. The methods are quite similar, since in the case base plans are stored according to the conditions where these plans were carried out.

In the case base plans are stored as a consequence of a situation composed by a series of elements. It cannot be a general planner; it has to be

related to a certain knowledge field. Different knowledge fields should have different implementations of the architecture. So, the plans stored in the case base have a series of related circumstances that determine the execution of that plan. The case base is organized according to the parameters determining the initial situation.

When a new situation is introduced into the system, it is included with the plan that solves that situation. The new situation will be placed close to similar situations stored in the case base. The cases must have a kind of metric factor in order to determine their position into the case base, and the proximity of the elements will be directly proportional to their similarity.

When a new situation arrives to the system, the most similar situations stored in the case base are retrieved. The solution to the situation will be an adaptation of the plans stored in the case base. If, there are any changes during the execution of the plan, or the plan fails, there should be mechanisms to modify the solution plan according to the changes produced.

5.7. SUMMARY AND CONCLUSIONS

The OBaMADE architecture represents an evolution in the concept of multiagent systems by introducing an organizational element among the agents and incorporating a CBR methodology as a reasoning core.

This chapter explained the OBaMADE architecture, describing the organizations that compose the architecture and the internal elements of all the existing organizations. Interfaces, communication and services were also explained, indicating the way all of them work in an individual way, as well as how they cooperate to achieve a common objective.

The external element of the architecture, the interface, was solved by the use of light interface agents that take the information given by the users or the systems where they are located, and send it to the system. The interfaces showed to the users are decided by the internal *interface*

organization, depending on the type of device used by the user and also depending on the type of request done .Once in the system, the information passes through the *communication organization* that processes the data, to determine what will be done with the information received. Depending on the type of communication established between the interface agents and the *communication organization*, the agents within the *communication organization* choose where to send the information and decide if a response is required from the internal organizations of the system. The *communication organization* chooses from the two *services organizations*, and sends to one of them the request received by the system.

The core of the system is composed by a Case-Based Reasoning group of services that encapsulate the CBR methodology. The implemented services cover the four main phases of the methodology and give solutions by reusing the stored information, extracting knowledge adapted to the problem to be solved. Those services are included in the *CBR services organization* where a set of agents solve the requests received from the *communication organization* and sends the response, if needed, back to the *communication organization*, which finally sends it to the user through the *interface organization.*

The *additional services organization* cover a series of needed services that do not necessary follow the CBR methodology and that are not directly related with the solution generation. Those services are important but, in terms of resource allocation, are not so crucial as the solved by the *CBR services organization*, where solutions are required and where the speed and reliability is higher than in this complementary organization.

The organizations of agents used to design the OBaMADE architecture represent an evolution of the multi-agent systems, where the structuring capabilities of the agents are taken to a higher extent by improving their socialization properties. The agents being part of those

organizations collaborate to obtain a common aim and share their objectives in a transparent way, without interfering with the normal dataflow within the systems.

In the next chapter, the results obtained with the OBaMADE architecture will be shown. Two case studies were chosen to apply the architecture. The application chosen to check the correction of the architecture was a prediction generation, where historical data is used to obtain new predictions to new problems.

"Advice is judged by results, not by intentions." *Cicero*

6. Application ~ Case Studies

The OBaMADE architecture has been successfully applied to two case studies. The first one involves the application of the OBaMADE architecture to the oil spill problem. To present the application of the architecture to the problem, the problem itself is first explained, including the methods for acquiring the data, the transformation, and the methods used to apply the technology to solve the problem. The second case study, which was applied to forest fires, served to more extensively check the OBaMADE architecture.

When a new architecture is created, it is necessary to apply it to solve the problems it is intended to solve. In this chapter, the application of the OBaMADE architecture to two different case studies is explained. While showing the application of the architecture, its prediction capabilities are shown. Those case studies are, both of them, located in natural environments, where different real-time parameters are involved and where different type of users interact with the systems at the same time, but playing different roles, interacting

among themselves and with the system in a concurrent way.

The first case study where the OBaMADE architecture was applied is the oil spill problem. When an oil spill occurs, the natural risks are evident and complicated decisions must be made in order to keep the risk from becoming a great natural disaster. The ability to predict if an area is going to be affected by the slicks generated after an oil spill will be highly useful in making those decisions.

The second case study to which the OBaMADE architecture was applied is the forest fire propagation prediction. This problem is similar to the first one analyzed, the oil spill. This second case study served as a validation procedure to check the correction of the architecture. The OBaMADE architecture was successfully applied to this second case study, generating a prediction consisting on the probability of finding fires in certain geographic area.

In both cases, the application of the OBaMADE architecture has generated quite optimistic results, predicting the future situation in a high degree of success.

6.1. OIL SPILL PREDICTION

The ocean is a highly variable environment where accurate predictions are difficult to achieve. The complexity of the modelling system is increased if external elements are introduced into the analysis. In this case, oil spill data is added to the inherent complexity of the ocean, generating a rough set of elements. To model an environment similar to what is obtained after adding oceanic variables, weather conditions and oil spills, it is necessary to measure different parameters such as wind, current, pressure, etc. To predict the presence or absence of oil spills in a certain area their previous positions must be known. That knowledge is provided by the analysis of satellite

images, from which the position and size of the slicks are obtained.

6.1.1. PROBLEM DESCRIPTION

After an oil spill, it is necessary to determine if an area is going to be contaminated or not. To determine the presence or absence of contamination in an area, it is necessary to understand the behaviour of the slicks generated by the spill.

First of all, the position, shape and size of the oil slicks must be identified. The most precise way to acquire that information is by using satellite images. SAR images are the most commonly used to automatically detect this kind of slick [Solberg *et al.*, 1999]. SAR images have been interpreted using CBR systems both for monitoring [Li and Yeh, 2004] or classification [Chen *et al.*, 2007] purposes. The satellite images show certain areas where there seems to be nothing, such as zones with no waves, that are in fact oil slicks. *Figure 20* shows a SAR image of a portion of the western Galician coast, as along with some black areas corresponding to the oil slicks. With SAR images it is possible to distinguish between normal sea variability and slicks.

It is also important to make a distinction between oil slicks and look-alikes. Oil slicks are quite similar to quiet sea areas, so it is not always easy to discriminate between them. If there is not enough wind, the difference between the calm sea and the surface of a slick is less evident, which may lead to and more mistakes when trying to differentiate between an oil slick and something that it is not a slick. This is a crucial aspect in this problem that can also be automatically performed by a series of computational tools.

Once the slicks are identified, it is also crucial to know the atmospheric and maritime situation that is affecting the slick at the moment that it is being analysed. Information collected from satellites is used to

obtain the atmospheric data needed. That is how different variables such as temperature, sea height and salinity are measured in order to obtain a global model [Stammer *et al.*, 2003] that explains how slicks evolve.

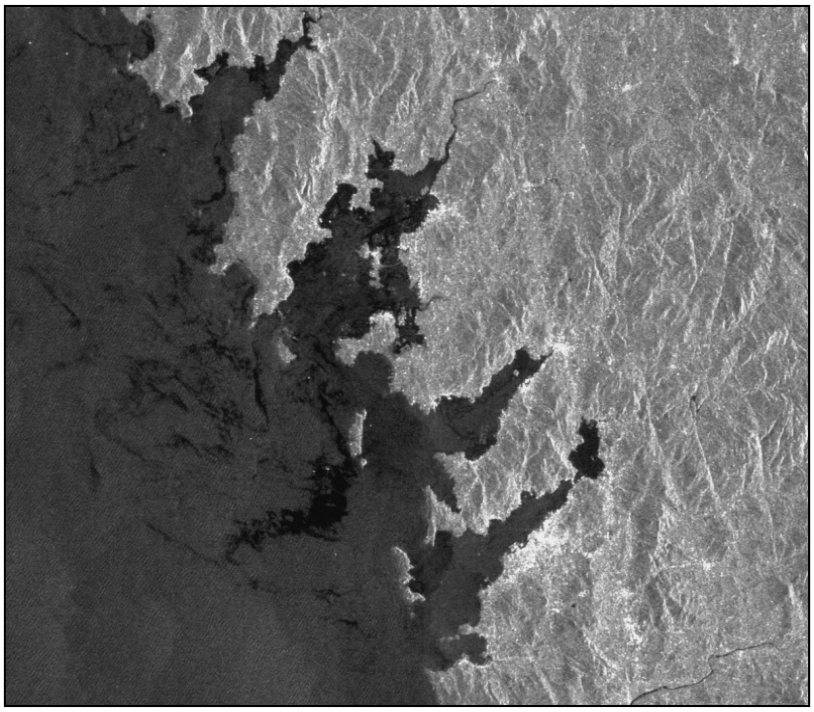


Figure 20. *SAR image of the north west of Spain, showing oil spills near the coastal zones.*

There have been different ways to analyze, evaluate and predict situations after an oil spill. One approach is by simulation [Brovchenko *et al.*, 2002], where a model of a certain area is created, introducing specific parameters (weather, currents and wind) and working with a forecasting system. Using this methodology, it is easy to obtain a good solution for a certain area [Elhakeem *et al.*, 2007], but it is quite difficult to generalize in order to solve the same problem in new zones. It is also possible to create a model for a specific and problematic area[Periáñez and Pascual-Granged, 2008], which is a great help, albeit limited, because it is not possible to

apply that same solution to different geographical areas. Current data must be considered in order to create contingency plans that could help to minimize environmental risks [Copeland and Thiam-Yew, 2006].

Another way to obtain a trajectory model is to replace the oil spill by drifters [Price *et al.*, 2003] comparing the trajectory followed by the drifters with the already known oil slick trajectories. If the drifters follow a trajectory similar to the one that followed the slicks, then a model can be created and there will be a possibility of creating more models in different areas. Another way of predicting oil slicks trajectories is to study previous cases to obtain a trajectory model for a certain area with different weather conditions [Vethamony *et al.*, 2007]. Another trajectory model is created to accomplish the NOAA standards [Beegle-Krause, 1999], where both the 'best guess' and the 'minimum regret' solutions are generated.

One step beyond the solutions previously explained are the systems that combine a major set of elements in order to generate response models to solve the oil spill problem.

A different view is given by complex systems [Douligeris *et al.*, 1995] that analyze large data bases (environmental, ecological, geographical and engineering) using expert systems. This way, an implicit relationship between problem and solution is obtained, but with no direct connection between past examples and current decisions. Nevertheless arriving at these kinds of solutions requires a great deal of data mining effort. Monitoring the spills [Benmecheta and Lansari, 2007] also gives a good quantity and quality of information, using the variety of techniques available [Qingling and Ying, 2007].

Once the oil spill is produced there should be contingency models that make a fast solution possible [Reed *et al.*, 1999]. Expert systems have also been used, whereby the stored information from past cases is used as a repository where future applications can find structured information. Other

complete models have been created, with the aim of integrating the different variables affecting the spills [Belore, 2005], always trying to get better benefits than the sum of the possible costs generated by all the infrastructure needed to respond to a generated problematic situation.

The final objective of all these systems is to become decision support systems that can help to take all the decisions that need to be taken in an organized manner. To achieve such a great objective, different techniques have been used, from fuzzy logic [Liu and Wirtz, 2007] to negotiation with multi-agent systems [Liu and Wirtz, 2005].

6.1.1.1. DETECTION

The first step in the solution of this kind of problems is to detect the oil spills in the ocean. There are different methods and techniques that can be applied to detect the slicks in the ocean. Most of them use information obtained from different satellites.

It is possible to detect oil spills by analyzing images generated by radiometers [Cai *et al.*, 2007], where the sea surface temperature is analyzed to determine where the oil slicks are. There are different kind of sensors used to remotely detect the presence of an oil spill, from visible sensors to satellite remote sensing (also using infrared, ultraviolet, radar, microwave and laser) [Jha *et al.*, 2008].

However, the most common images used to determine the presence of oil spills are SAR images SAR [Solberg *et al.*, 2007], where different techniques have been applied to distinguish the oil slicks. The main objective is to create systems that may detect the slicks in an automatic way [Keramitsoglou *et al.*, 2006, Tello *et al.*, 2006]. Other investigations use supervised methods or partially supervised methods to create systems that may detect oil spills [Montali *et al.*, 2006, Mercier and Girard-Ardhuin, 2006].

It is very important to discriminate between oil spills and look-alikes, so as to not generate unnecessary alarms [Topouzelis *et al.*, 2007]. Finally, it is also possible to monitor the ocean and the evolution of the oil spills by using satellite data [Cotton, 2007, Nelson *et al.*, 2006].

6.1.1.2. RESPONSE

Once the spill has been produced, it is crucial to generate quick and accurate responses to minimize the environmental damages created by the spill.

Data about the ocean currents must also be considered in order to create contingency plans that could help to minimize environmental risks [Copeland and Thiam-Yew, 2006]. Specific models can be created for special geographical zones, where the oceanic behaviour is quite unusual [Periáñez, 2007]. If an oil spill is produced, it is important to analyze the response given to a specific situation [Tuler *et al.*, 2006] in order to improve possible future accidents by discovering faults and avoiding mistakes.

Monitoring the dangerous geographical areas can be a great help to create models that can evaluate the various possibilities in which the situations can evolve. This monitoring process can be carried out by using different techniques [Benmecheta and Lansari, 2007, Qingling and Ying, 2007].

If by chance there are no accidents to monitor or to use, simulations can generate useful information that can be used for future situations [Wirtz *et al.*, 2007]. When performing a simulation, natural conditions are reproduced and the accident is substituted by artificial elements that attempt to model the real evolution of the slicks.

6.1.1.3. FORECASTING

Perhaps the most difficult task when treating natural information related with dynamic environments is to forecast their evolution. The ocean is a complex environment and predicting the evolution of oil spills (an artificial agent added to the water) is a complicated task.

Hybrid models can forecast trajectories and evaluate possible risks after an oil spill [Jordi *et al.*, 2006]. Those models integrate different techniques to try to reduce the damage caused by the spills. Combining wind driven drifts and climactic variables can produce a robust forecasting model [Carracedo *et al.*, 2006]. Drifts simulate the actual evolution of the oil slicks in the ocean, as their movements are mostly driven by wind, at least in open ocean. To predict the evolution in specially complicated areas, a specific model can be created for that geographical area, to simplify the generation of results [Periáñez and Pascual-Granged, 2008].

Finally, it is important to know what happens when an error is produced in systems such as those that have been previously explained [Jorda *et al.*, 2007]. It is important to know the effects that an error will introduce into both the system and the predictions in order to help solve future problems in real situations.

6.1.2. DATA USED AND APPLICATION OF OBAMADE

To evaluate the correction of the application of the OBaMADE architecture to the oil spill problem, a series of historical data taken from the Prestige accident were used. The solution proposed in this study generates the probability (between 0 and 1) for different geographical areas of finding oil slicks after an oil spill. The proposed system was constructed

using historical data and checked by using the data acquired during the Prestige oil spill between November 2002 and April 2003. Most of the data used to develop the proposed system was acquired from the ECCO (Estimating the Circulation and Climate of the Ocean) consortium [Menemenlis et al., 2005]. The position and size of the slicks was obtained by using SAR (Synthetic Aperture Radar) satellite images [Palenzuela et al., 2006].

Table 1. *Variables used in the oil spill problem.*

VARIABLES	DEFINITION	UNIT
Longitude	Geographical longitude	Degree
Latitude	Geographical latitude	Degree
Date	Day, month and year of the analysis	dd/mm/yyyy
Sea Height	Height of the waves in open sea	m
Bottom pressure	Atmospheric pressure in the open sea	Newton/m^2
Salinity	Sea salinity	ppt (parts per thousand)
Temperature	Celsius temperature in the area	ºC
Area of the slicks	Surface covered by the slicks present in the analyzed area	Km^2
Meridional Wind	Meridional direction of the wind	Degree
Zonal Wind	Zonal direction of the wind	Degree
Wind Strenght	Wind strength	m/s
Meridional Current	Meridional component of the ocean current	m/s
Zonal Current	Zonal component of the ocean current	m/s
Current Strenght	Ocean current strength	m/s

Table 1 shows the parameters used to create the case base that will provide the data used to solve new problems. Past solutions are stored in the system, in the case base. In the system created, the cases contain information about the oil slicks (size and number) as well as atmospheric data (wind, current, salinity, temperature, ocean height and pressure). The system generated combines the efficiency of the CBR systems with artificial intelligence techniques in order to improve the results and to better generalize from past data.

The system developed determines the probability of finding oil slicks in a certain area. To generate the predictions, the system divides the area to be analyzed into squares of approximately half a degree per side. The system then determines the number of slicks present in a given square. The squares where the slicks are located are coloured with different gradation depending on the quantity of the squared area covered by oil slicks.

The squared zone determines the area that is going to be analyzed independently. The values of the different variables in a square area at a certain moment as well as the value of the possibility of finding oil slicks on the following day is called a case, which defines the problem and proposes the solution.

The parameters used in this case studied will now be explained in detail:

- *Longitude and Latitude:* it is crucial to know the position where an oil slick is located. But it is also important to decide in which direction the slicks are going to move. The position itself it is not as critical in determining the final result, at least in open ocean, where there do not are any specific models determined by the variations of the coast.

- *Date:* this is an important element as it establishes the temporal relationship between past situations (problems) and future situations (solutions) for the same location.
- *Sea Height, temperature bottom pressure and salinity:* atmospheric and weather parameters that may help the neural networks used in the reuse phase to enrich the solution proposed.
- *Area of the slicks:* represents the proportion of the square area affected by the oil slicks. It is an important parameter because it represents the evolution of the slick in the area. If this parameter increases its value, it indicates that new slicks are coming from neighbouring areas. If its value decreases, then the slicks in this area are moving to other neighbouring areas.
- *Wind:* an important element, as it is the most responsible for the movement of the slicks. The wind is divided into three components, meridional (the component of the wind parallel to one meridian), zonal (the component of the wind parallel to one parallel of latitude) and strength (representing how strong is the wind).
- *Current:* like the wind, it is also important for determining the movement of the slicks. It is also divided into three components, following the same structure of the wind components.

6.1.3. RESULTS

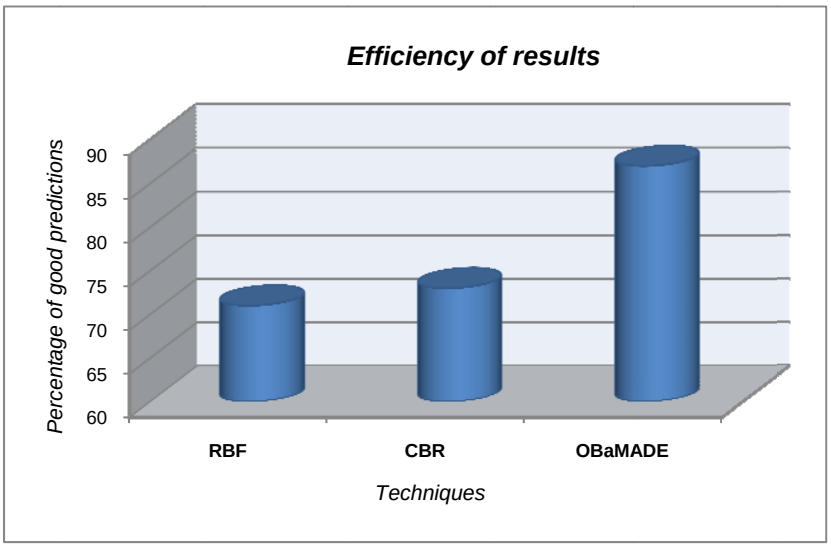
The data used to train the system were obtained from different satellites. Temperature, salinity, bottom pressure, sea height, number and area of the slicks, along with the location of the squared area and the date were all used to create a case. All these data define the problem case and also the solution case. The solution to a problem defined by an area and its variables is the same area, but with the values of the variables changed according to the prediction obtained from the CBR system.

The WeVoS algorithm has proved to be more efficient than other existing algorithms used to organize, classify and visualize information [Baruque and Corchado, 2007]; it has obtained better results than simple ensembles of SOMs, fusion Euclidean Distance, Voronoi Polygon Similarity and Ordered Similarity. The main feature of this novel algorithm is the reliable visual representation of the dataset, which is measured by the distortion, rather than the classification accuracy or the reduction of the quantization error; thus maintaining the topology preservation feature, which is one of the most important for the original model that it is intended to improve.

When the developed system was used with a subset of the data that had not been previously used to train the system, it produced quite optimistic results. The predicted situation was contrasted with the actual future situation. The future situation was known, as past data was used to train the system and also to test the correction of its results. In most of the variables, the proposed solution had an accuracy rate of nearly 90%. When using the system created with the OBaMADE architecture, the efficiency of the results was better than what was obtained by using previous and simpler applications; those improvements can be seen in the figures shown in this section.

In *figure 21*, the system results are compared with those obtained with two other systems. The first one, *"RBF"*, is a simple RBF network, where data is introduced by training the network, and the results are obtained by a generalized application of the information internally stored in the network. The *" Basic CBR"* system represents a CBR system applied to forecast oceanographic methods [Corchado and Aiken, 2002]. This system uses neural networks, specifically the Radial Basis Function network, during the adaptation process of the recovered cases. The neural network has a process for recovering elements from a network knowledge base,

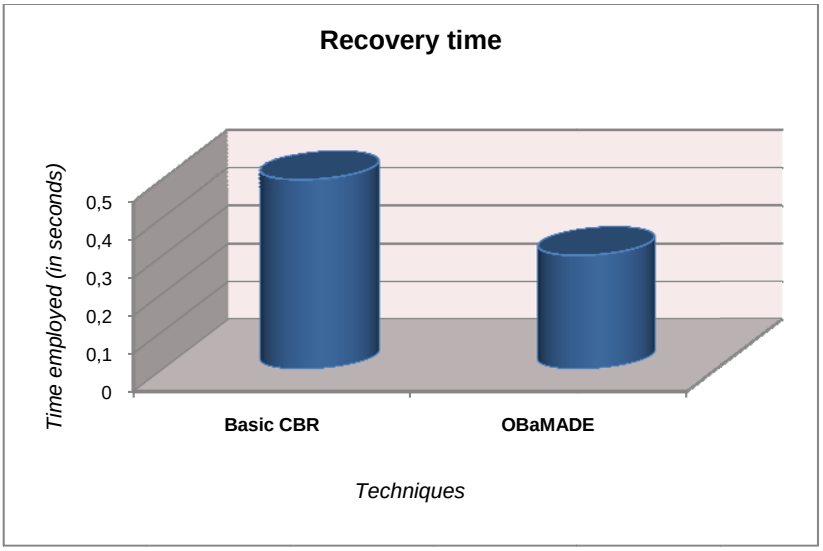
from where the neural network retrieves the parameters to calibrate the network. This CBR system has been applied to oceanographic problems. As can be seen in *figure 21* the proposed system is better than the other systems.



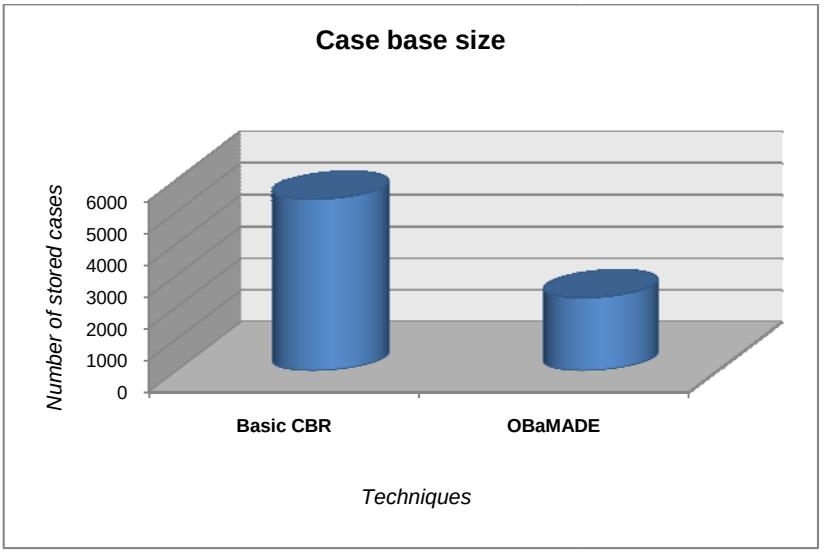
ral

Figure 22 compares the system developed under the OBaMADE architecture with the *"Basic CBR"* previously explained, in terms of the time required to recover the most similar cases from the case base. As shown, the new system is better than the basic one.

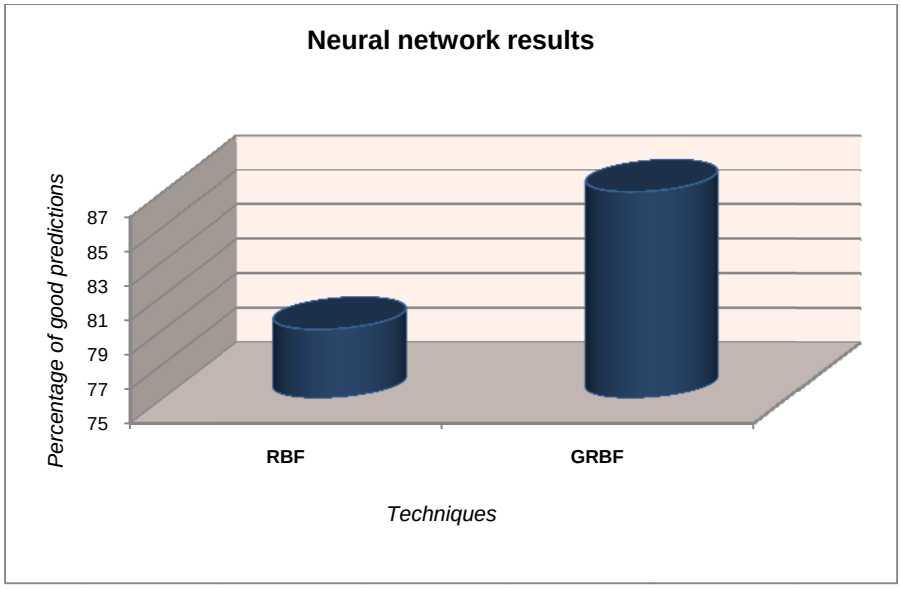
In *figure 23* there is a comparison between the size of the *"Basic CBR"* where no analysis of the store information is done, and that of *"OBaMADE"* where data is structured and analyzed before being stored. *Figure 24* provides a comparison between the results obtained in the reuse phase by a *"RBF"* neural network, which represents the classic version of this network, and the *"GRBF"*, where the growth of the neural network is progressive and adapted to the data.



' that of



d that of



eural

For each problem defined by an area and its variables, the system offers nine solutions: the same area, with its proposed variables, and the eight closest neighbours. This type of prediction is used in order to clearly observe the direction of the slicks, which can be useful in determining the coastal areas that will be affected by the slicks generated after an oil spill. The proposed solution does not generate a trajectory, but a series of probabilities in different areas, which is far more similar to the real behaviour of the oil slicks.

Table 2 shows a summary of the results obtained in which different techniques are compared. The table shows the evolution of the results along with the increase in the number of cases stored in the case base. The results for each of the techniques being analyzed improved when the number of cases stored was increased. The *"RBF"* column represents a simple Radial Basis Function Network that is trained with all the available

data. The network gives an output that is considered a solution to the problem. The *"Basic CBR"* column represents a pure CBR system, with no additional techniques included. It is the *"Basic CBR"* described before. The *"GRBF + CBR"* column corresponds to the possibility of using a GRBF network combined with a simple CBR system. The recovery from the CBR is achieved by using the Manhattan distance to determine the closest cases to the introduced problem. The RBF network works in the reuse phase, adapting the selected cases to obtain the new solution. The results of the *"GRBF+CBR"* column are normally better than those of the *"CBR"*, mainly because useless data are eliminated prior to generating the solution. Finally, the *"OBaMADE"* column shows the results obtained by the proposed system, which are better still than the three previous solutions analyzed.

Table 2. *Percentage of good predictions obtained with different techniques – Oil spill problem.*

NUMBER OF CASES	RBF	BASIC CBR	GRBF + CBR	OBaMADE
100	45 %	39 %	42 %	43 %
500	48 %	43 %	46 %	46 %
1000	51 %	47 %	58 %	64 %
2000	56 %	55 %	65 %	72 %
3000	59 %	58 %	68 %	81 %
4000	60 %	63 %	69 %	84 %
5000	63 %	64 %	72 %	87 %

Table 3 shows a multiple comparison procedure (*Mann-Whitney* test) used to determine which models are significantly different from the others. The asterisk indicates that these pairs show statistically significant differences at the 99.0% confidence level. *Table 3* shows that the

OBaMADE system presents statistically significant differences compared to the other models.

Table 3. *Multiple comparison procedure among different techniques.*

	RBF	CBR	GRBF + CBR	OBaMADE
RBF				
CBR	✱			
GRBF+CBR	=	=		
OBaMADE	✱	✱	✱	

6.2. Fire Propagation Prediction

The second case study is presented here. The OBaMADE was also applied to predict the evolution of forest fires, considering the areas that could be eventually affected by the fires.

The structure of this subsection is similar to the previous one. First the problem will be introduced, describing the main characteristics of this kind of problem and also a brief revision of the different techniques and systems used to solve this problem. Then, the data used to check the OBaMADE architecture is described, and finally the application of the architecture and the results obtained are shown.

6.2.1. Problem Description

Forest fires are a very serious hazard that, every year, cause significant damage around the world from an ecological, social, economical and human point of view [Long, 2001]. These hazards are particularly dangerous when meteorological conditions are extreme with dry and hot seasons or strong wind. For example, fire is a recurrent factor in Mediterranean areas.

Fires represent a complex environment, where multiple parameters are involved. In this sub-section, a series of applications and possible solutions are explained. They are different approaches to the forest fire problems, including all the main phases existing in the evolution of this kind of problem.

The causes that produce forest fires are many, and the great majority are related with one or another form of human factors (more than 90% of forest fires are provoked by human action); in addition, fires in degraded forests are worse than those that occur in more intact forests [Cochrane, 2002].

6.2.1.1. DETECTION

Detection is the first step, it is necessary to detect where a fire has started, in order to act as quickly as possible. So, detection systems and techniques are crucial to quickly determine where the fire is and to fight against it. There have been multiple ways and systems of detecting forest fires. Some of these will now be described.

Some techniques, previously applied to the monitoring of major natural and environmental risks, have been transformed to forest fire detection [Mazzeo *et al.*, 2007]; in this case it is a multi-temporal robust satellite technique (RST). This system uses AVHRR MIR images, to detect the fires.

MODIS (Moderate Resolution Imaging Spectro-radiometer) offers high quality images that have been used to which a detection algorithm is applied [Giglio *et al.*, 2003]. It allows the detection of small fires and the reduction of false alarms. False alarm reduction can also be done by infrared forest fire detection [Arrue *et al.*, 2000]. In this case artificial vision, neural networks and expert fuzzy rules are combined to reduce the number of false alarms in that kind of image

analysis. Satellite images can also be used to detect forest fires analyzing those images (NOAA/16-AVHRR) with a perceptron neural network [Muñoz *et al.*, 2007].

Black and white cameras can also be used to obtain an autonomous fire detection [Den Breejen *et al.*, 1998]. In this case, images are compared, and if something new appears, it is analyzed to check if it is a smoke plume. If it is, an alarm is sent and the process to fight the fire begins.

It is also important to check the correction of simulations, which is possible to do when there is a large enough quantity of data available [Damoah *et al.*, 2004]. In this case the simulation models have been compared with a real smoke transport situation, where the smoke plumes circumnavigated the globe in seventeen days.

Finally, animals were also used to carry specific sensors (thermo and radiation sensors with GPS features) so that they can serve as Mobile Biological Sensors to detect forest fires [Sahin, 2007].

6.2.1.2. PREDICTION

Forest fires can be *estimated,* as a kind of prediction, by using a fuzzy system to create decision support systems for a forest [Iliadis, 2005]. Parallel computing has also been applied to the prediction problem in this knowledge field. In this occasion an adaptive system could help to generate predictions by changing at the same time that the environment changes [Rodríguez *et al.*, 2008].

The spread of the fire highly depends on whether parameters [Martins Fernandes, 2001]. Simulating variations on the parameters, it is possible to determine the evolution of the fire in different conditions.

One easy way to analyze the great amount of data generated in such environmental related problems is to divide the data into smaller pieces [Brillinger *et al.*, 2003]. The results obtained with the smaller elements may be generalized to obtain future predictions.

Graphical models and interfaces help to create realistic models and simulations [Serón *et al.*, 2005]. The existence of a graphical representation makes it easier for experts to introduce their knowledge into the systems.

Statistics are a great help in predicting problems. If more than one solution is considered, the probabilities of being in the right path increase. [Bianchini, 2006]. If no possibility is rejected, then the scope is bigger, but also the potential amount of data available for further analysis.

6.2.1.3. MODELS AND SYSTEMS

As stated in the description of the existing applications for solving the oil spill problem, models and systems represent the most evolved situations, offering the most complex solutions and involving the highest number of elements.

Simple models can be generated by using automata [Karafyllidis and Thanailakis, 1997], representing the spread of the fires according to the states of the automata, and adapting their evolution to external parameters. Mathematical models can also be applied but with a more complicated introduction of the external parameters (such as weather conditions) into the models created [Montenegro *et al.*, 1997]. Decision support systems are one of the first high level approaches to this kind of problems [Wybo *et al.*, 1998]. They normally use different sources of information to generate decisions based on the variety of data available.

6.2.2. DATA USED AND APPLICATION OF OBAMADE

The data used to check the validity of the OBaMADE architecture was applied to the forest fire problem. The data used is part of the SPREAD project [Spread, 2004], in particular the Gestosa field experiments that took place in 2002 and 2004 [Gestosa, 2005]. The experiments of the Gestosa field began in 1998 and were completed in December 2004. They aimed to collect experimental data to support the development of new concepts and models, and to validate existing methods or models in various fields of fire management. The study area is located in Central Portugal (Gestosa, 40º 15`N, 8º 10' O) in a hillside of the Serra de Lousa, whose altitude is between 800 and 950m above sea level.

To safeguard the safety of the burns and to carry out different sorts of tests and measurements, the terrain was divided into dedicated plots with regular shapes and dimensions separated by firewalls to limit the spread of the fire and to keep it inside the desired boundaries during each burn. Those experimental burning plots were established in forest service lands, in the Gestosa forestry perimeter. In general, these experimental plots are located together in the same vegetation mosaic, which contains shrubs and some isolated *Pinus pinaster* trees. Three arboreal species are dominant in the area: *Erica umbellate, Erica australis* and *Chamaespartium tridentatum.*

The application of OBaMADE to this new problem followed the same process as the application to the oil spill problem. First, the areas analyzed were divided into squares, where meteorological parameters were measured and registered. All of the data obtained are used to create the case base and train the neural networks. On this occasion the data used are shown in *table 4.*

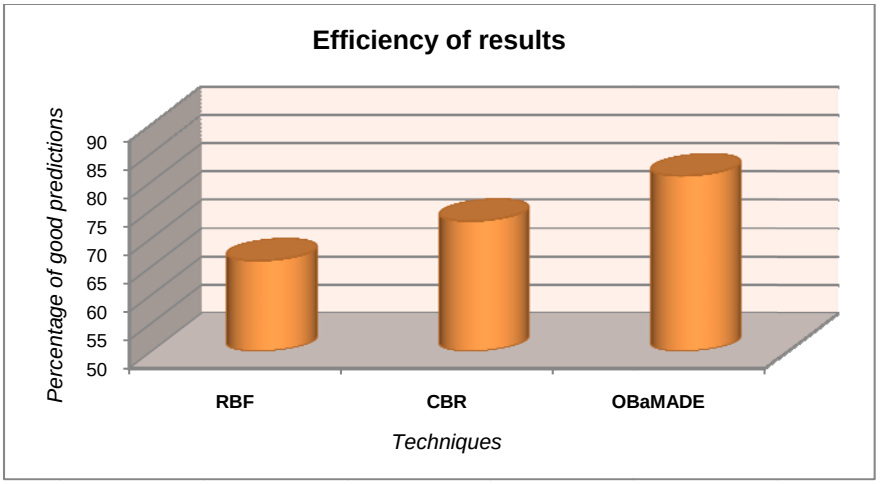
Table 4. *Variables used in the forest fire problem.*

VARIABLE	DEFINITION	UNIT
Longitude	Geographical longitude	Degree
Latitude	Geographical latitude	Degree
Date	Day, month and year of the analysis	dd/mm/yyyy
Bottom pressure	Atmospheric pressure in the open sea	Newton/m^2
Temperature	Celsius temperature in the area	ºC
Area of the fires	Surface covered by the fires present in the analyzed area	Km^2
Meridional Wind	Meridional component of the wind	m/s
Zonal Wind	Zonal component of the wind	m/s
Wind Strenght	Wind strength	m/s

As shown in *table 4*, most of the data used in this problem are the same as in the oil spill problem. In fact there are some parameters that are not present here, as the problem is located on land and not in open sea. Nevertheless, the variability and complexity of the problem is high; the wind conditions can change faster in forest lands than in open ocean and the variability of the temperature is also higher, which implies a smaller reaction time limit in order to adapt to the changes. The combination of natural parameters and predictions needs make it more complicated to be accurate.

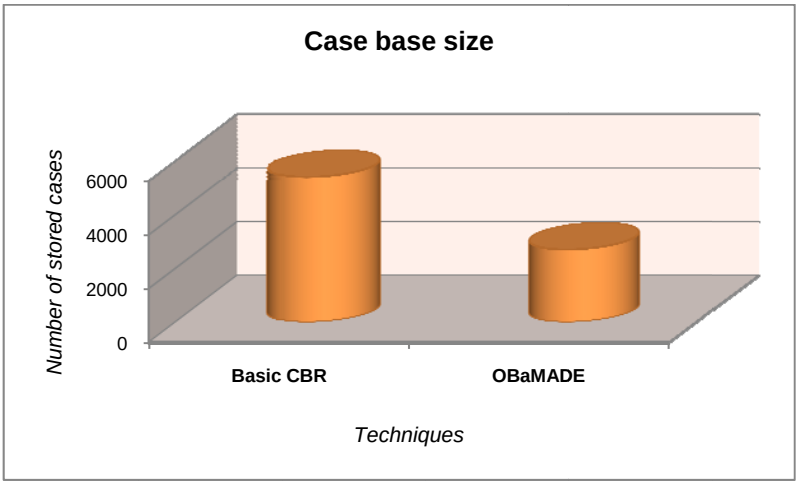
6.2.3. RESULTS

The experiments and comparisons performed with the forest fire problem are equivalent to those performed with the oil spill problem. A summary of the results of those experiments will be presented, focusing on the size of the case base and on the efficiency of results, the response time, and the results of the neural network.



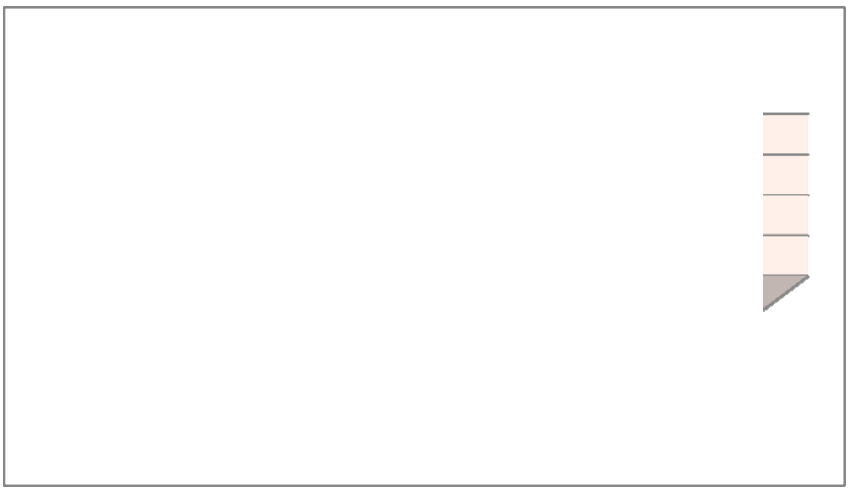
ıetwork
tudy.

Figure 25 shows the improvement obtained with the system based on the OBaMADE architecture, compared with *the "Basic CBR"* explained before, when developing the results of the oil spill problem. The results are quite better with the new application; the use of specific neural networks and an effective communication improves the overall results of the system.



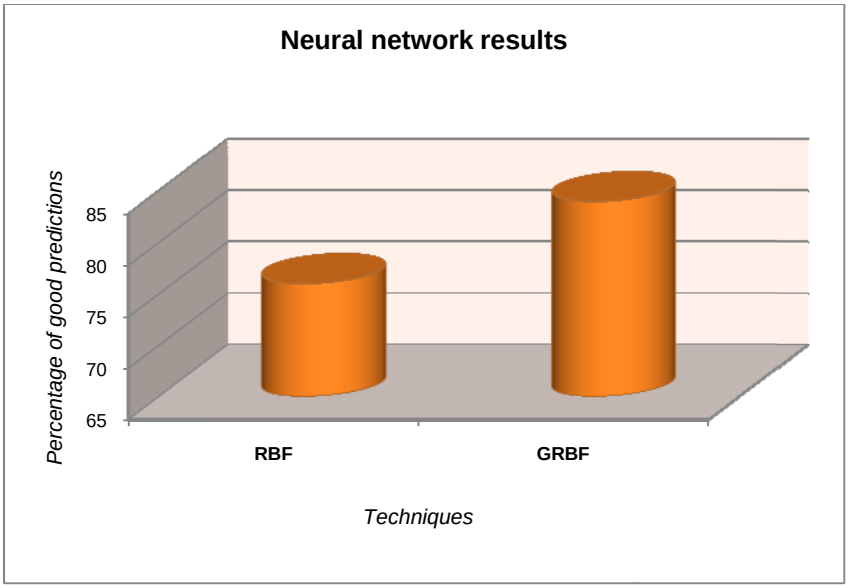
hat of

Figure 26 shows the reduction of the case base produced by organizing the case base with a specific neural network system. Reducing the number of parameters to store and organizing the stored information generate a great reduction in the size of the case base, which is also crucial for the results shown in *figure 27*, where a comparison of the recovery time is seen. If the case base is smaller and better organized, then the recovery time and effort needed to retrieve the most similar cases from the case base is much quicker.



ıat of

Finally, *figure 28* shows the improvement obtained by using a *GRBF* network to the reuse phase instead of the classic *RBF*. This new adaptation of the RBF standard is better suited for the inner structure of the case base, which implies an improvement of the results. If both the data used to train the neural network and the neural network itself share a common inner philosophy of growing, the final result can be better than without avoiding that equivalence.



eural
1 to the

Table 5 shows the comparative results obtained. The techniques used for comparison are the same as in the oil spill case study. As occurred in the previous case study, the results are better when the quantity of information stored in the case base is higher. The increase in the number of cases stored improves the results, having the possibility of recovering *better* cases from the case base to use them to generate the predictions. The results are better when applying a system created under the OBaMADE architecture than using other *simpler* techniques. That quality of the results is quite hopeful and creates quite optimistic spectations to apply the OBaMADE architecture to other case studies, to other knowledge fields and to other kind of problems related with the distributed environments. OBaMADE shows here, again, its prediction capabilities, being successfully able to generate accurate predictions to a real-life problem in a complex environments.

Table 5. *Percentage of good predictions obtained with different techniques – Forest fires problem.*

NUMBER OF CASES	RBF	BASIC CBR	GRBF + CBR	OBaMADE
100	43%	37%	43%	45%
500	46%	42%	48%	50%
1000	52%	44%	56%	66%
2000	57%	53%	66%	75%
3000	59%	56%	69%	82%
4000	62%	60%	71%	86%
5000	64%	62%	72%	90%

6.3. SUMMARY AND CONCLUSIONS

It has been demonstrated that the presented architecture represents an evolution of previous existing techniques. It could be applied to different kinds of problems, offering great adaptation and generalization capabilities. It is a flexible architecture, capable of generating solutions to different kinds of problems in a great variety of situations.

The two case studies presented in this chapter prove the theoretical improvements predicted in the previous evaluation. The evolution of the situation of the oil spills in some geographical areas can be predicted by reusing historical data stored in the inner case base. Past information is used to solve new problems. Previous evolution data related with the oil spills is reused to generate new solutions. Different neural networks are used both to organize the case base and to generate the solution. The organization of the case base through a neural network improves the recovery time and makes it possible to employ a smaller number of cases that are more useful. The use of neural networks in the reuse phase generates great results by adapting the

retrieved cases to generate the new solution to the new problem.

ase

The same process is carried out with the forest fire case study. Historical data is stored, changing the parameters and adapting the architecture to the new problem. The system is requested to make predictions in which the cases are also structured in the case base, and retrieved to generate the proper solutions. The positive results obtained with the forest fire problem confirm the correction of the results obtained with the oil spills.

Figure 29 graphically shows the evolution of the results in the two case studies analyzed in this chapter. It can be clearly seen how the accuracy of the results improved while the case base size grow. At the same time, it is important to pay attention to the results obtained applying the OBaMADE architecture, that are always better than with the rest of the techniques analyzed for comparison.

In the next chapter, the model proposed will be analyzed from a more abstract point of view. Some conclusions to the work presented in this document will also be presented, while evaluating the process of the creation of this architecture. Some future possibilities of the OBaMADE architecture will also be presented, including new possible investigations that can be performed by applying this architecture without requiring a great amount of changes.

"The best way to predict the future is to invent it." *Alan Kay*

7. Architecture Evaluation and Conclusions

The OBaMADE architecture presented in this document is evaluated here, and the model represented by this architecture is analyzed, comparing it to other possible approaches to the distributed environments problem. After considering the study in its entirety, some conclusions and future research are explained, indicating the expected evolution of the architecture, and its possible future applications.

Prior to this chapter, the OBaMADE architecture was presented and explained. A complete state of the art of the technologies and methodologies used in this architecture were performed, both in previous chapters of this document and in the appendices. The results obtained applying the OBaMADE architecture were also shown in the previous chapter, analyzing the results obtained after applying the OBaMADE

architecture to two case studies.

The architecture proposed in this document achieves the main objectives that were initially proposed and improves previous approaches to solve this kind of problems. OBaMADE also uses case-based reasoning as the methodology for generating solutions to the different problems to which it may be applied. The CBR methodology makes great use of the information available. Past information is used to solve new problems, as with the two case studies presented in this chapter. Past solutions to past problems are used and adapted to solve new problems.

The OBaMADE architecture integrates the advantages of the multiagent systems, allowing it to solve similar problems. Structuring the agents of the architecture into organizations adds organizational capabilities to the architecture and makes it easier for the different parts of OBaMADE to communicate. Organizations allow the architecture to divide the different groups of agents according to their respective functionality and objectives. Being divided into groups (organizations) with the same common objectives, the communication between the organizations is easier, as only one agent in each organization is in charge of the communications tasks, reducing the complexity of the remaining agents.

This chapter will analyze and compare the OBaMADE architecture with other techniques usually employed to solve distributed environment problems. The advantages and disadvantages of the different techniques compared are also explained. It will also present the final conclusions of the investigation, showing the achievement of the initial objectives explained in the introduction of this document. Finally, some future lines of work and possible evolutions of the architecture are presented, introducing some possible new applications of the architecture to new knowledge fields, based on the main characteristics of the architecture.

7.1. THEORETICAL MODEL EVALUATION

OBaMADE represents an evolution of the existing models and architectures that have been solving the problems generated in distributed environments in recent years. Nevertheless, there are some important differences between the OBaMADE architecture and other models of distributed architectures. For the development of OBaMADE a balance between decentralization and intelligence was achieved. Decentralization is defined as the result of distributing the functionalities. A reuse feature can be obtained with this distribution and the independence that exists in the programming languages.

Intelligence is defined here as the result of the reasoning capabilities and the ability to adapt the behaviour in an autonomous way, and the ability to perceive stimulus from the context and react to them in a personalized way. While OBaMADE tries to achieve a balance between intelligence and decentralization, alternatives like SOA or Web Services present important limitations regarding the level of negotiation between services and context sensibility. CORBA is not sufficiently independent from programming languages, and the developed applications are not always compatible.

Finally, although multiagent platforms can provide quite useful tools to obtain intelligent systems, they do not facilitate the compatibility between platforms, nor do they offer the needed tools to obtain a more efficient decentralization of functionalities. *Figure 30* graphically shows the differences between the models explained here, showing the benefits of the OBaMADE architecture, compared with the other techniques explained here. The balance offered by OBaMADE between *intelligence* and *decentralization* is what makes it effective and able to be applied to different scenarios.

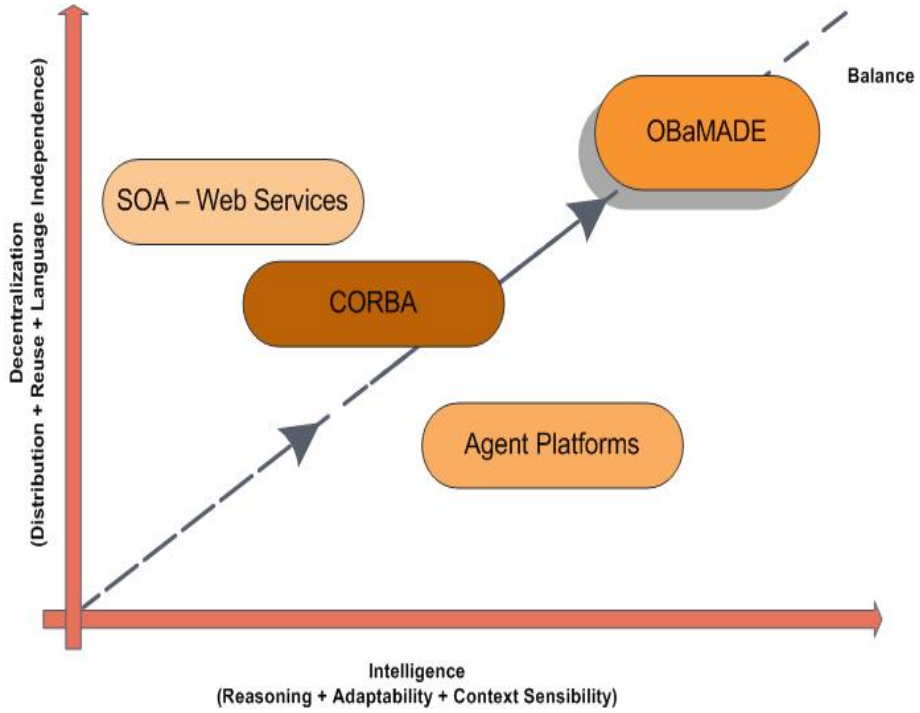


Figure 30. *Graphical comparison between OBaMADE and other architectural models.*

CORBA has some well known problems, as the complexity of being, at the same time, language-independent, platform-independent, suitable for distributed-systems development and maintaining backward compatibility. There are some interface problems between versions. Error handling is not extensible. The synchronization between client and server is crucial and not always well solved. Most of these problems are solved by open multi-agent based systems, such as organization based systems.

Optimal utilization of SOA requires additional development and design attempts as well as infrastructure which translate into costs escalation. When it comes to applications, Web Services and Service Oriented Architecture is not recommended for applications in which one way asynchronous communication is necessary, and where loose coupling is

considered undesirable and unnecessary. It is also not a good solution for homogenous application environments, like, for instance, an environment wherein all applications were built utilizing J2EE components. In these instances, it is not a good idea to introduce XML over HTTP for inter-component communications rather than utilizing Java remote method invocation. And, finally, for applications that need GUI based functionality it is not a proper solution. Like, for instance, a map manipulation application that has lots of geographical data manipulation. Such an application is not suited for heavy data exchange that is service based.

7.2. MODEL ANALYSIS

As explained in the previous chapter, the OBaMADE architecture improves both the theoretical and practical results of the existing architectures dedicated to distributed environments. Now it is time to compare the architecture with previous versions of the systems, analyzing the advantages acquired by transforming applications from local to distributed, and by integrating the organizations of agents and their services.

The performance of the OBaMADE architecture was compared with a previous local version of the system, with the same artificial intelligence techniques implemented, but without the use of agents and services. The tests performed consisted of the execution of the same series of predictions in both systems. There were 50 different problems to be solved. The executions were divided by introducing 1, 5, 10, 20, 30 or 50 requests at the same time. These executions were done 50 times and in the OBaMADE version, there were 5 different agents for each the type of problem.

Figure 31 shows the average time needed by the two systems to execute a series of requests. OBaMADE was able to improve the results obtained with the local version of the system. For small workloads in a local system, having no agents and communication involved can even be a little

faster than a distributed system with agents involved. But when facing higher workloads, the distributed approach shows its capabilities and is much quicker.

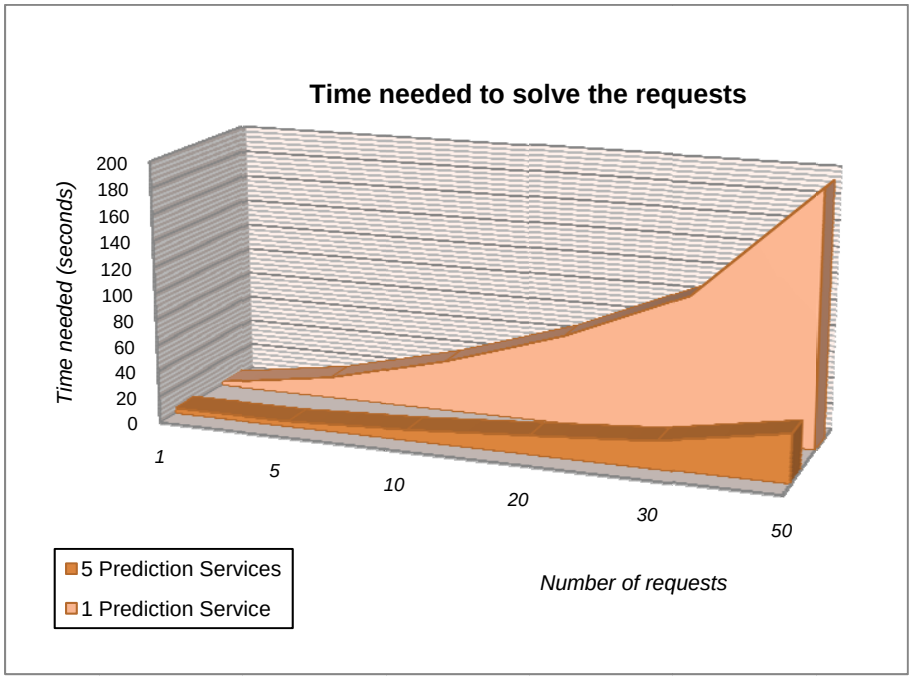
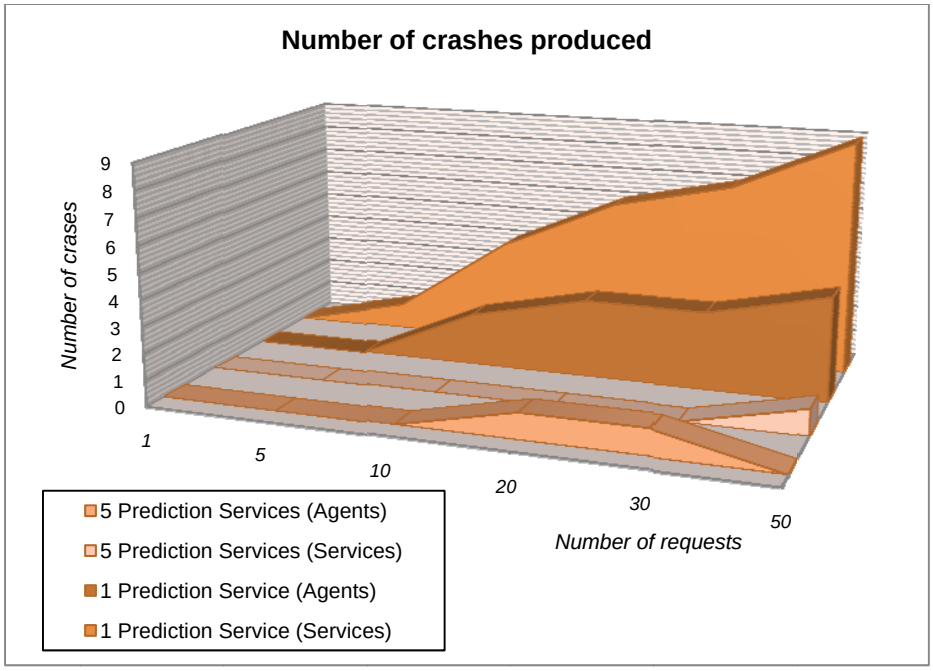


Figure 31. *Time needed to solve the requests by just one service or by five services simultaneously.*

One of the main contributions of OBaMADE to the problems related with distributed environments is the improvement of the performance and the robustness of the system. *Figure 32* shows the number of agents that crashed during the simulations. OBaMADE generates a smaller number of crashes and provides a bigger recovery capacity from crashes.

It is important to indicate that even after the great work done to reduce or even eliminate the crashes from the OBaMADE architecture, they do still occur, and it is then important to continue working to obtain the best possible results. Nevertheless, the results obtained up to now are quite hopeful.



of the

Any new development faces advantages and disadvantages. It is necessary to analyze them and improve the week aspects of any system. *Table 6* presents the main advantages and disadvantages of OBaMADE. In that table, the elements that make OBaMADE powerful are analyzed, and also the elements that may be improved in future developments to make the architecture better.

The OBaMADE architecture is still being developed, but preliminary results show that it is possible to apply it to complex systems, as the used before as case studies. Those case studies represent only one of the main fields where OBaMADE can be applied.

Table 6. *Advantages and disadvantages of the OBaMADE architecture.*

ADVANTAGES	DISADVANTAGES
✓ Optimization of the use and distribution of the resources. ✓ Programming languages independence. ✓ Services and applications support the computational effort. ✓ Facilitates the reuse of the functionalities. ✓ It has been successfully applied to two different case studies. ✓ Defines a set of technologies and methodologies that can be used in future similar developments.	✓ It is still under development and it is not fully debugged or formalized. ✓ It depends on agents platforms. ✓ It has only been applied to two different case studies. It is necessary to implement to different scenarios. ✓ There have not been applied standardized evaluations.

7.3. CONCLUSIONS

In this section the achievement of the objectives defined for this investigation is described, and it evaluates the initial hypothesis of the study: "*develop an architecture to solve problems related with distributed environments. The architecture should face those problems offering different interfaces to different users with different devices in a transparent way. The architecture has to be based in organizations of agents. The agents that make those organizations must be designed as dynamic agents. The agents being part of the inner organizations, which are in charge of the generation of the solutions, should incorporate reasoning mechanisms based on the Case-Based Reasoning methodology*". Within the framework defined by this research project the ability of the OBaMADE architecture to solve different problems in different scenarios has been tested, with a high scalability and reuse of resources. Thus, the architecture has demonstrated to be able to extract and model the functionalities of the agents as individual services, creating lighter

agents. Because it is more adaptable on execution time, the distributed approach given to the architecture makes it more flexible and failure tolerant. The OBaMADE architecture, presented in this paper, achieves some important objectives with regards to the fields related with this investigation.

OBaMADE provides a robust framework flexible enough to cover the requirements of systems designed to solve distributed environment problems. Dynamic scenarios with a great user interaction are properly solved by the architecture. Its computing elements work in a distributed way, collaborating to obtain a common result.

The core of the system is formed by a CBR set of services integrated with some artificial intelligence techniques designed to extract knowledge from the information. The agents within the architecture are part of organizations that communicate with each other to obtain a common objective and to make the proper decisions.

One of the lasts additions in the evolution of the OBaMADE architecture was its logo, which is shown in *figure 33*. It is also important to have a graphical representation of the architecture, and the one chosen to represent OBaMADE is a dynamic one, with the colours chosen to show the figures all through the document, with a graphical design inspired in the coordination of the elements that are part of OBaMADE and with a

typography adapted to that inspiration.

Lighter agents make it possible to expand the possibilities of development of applications based on the OBaMADE architecture to devices that do not necessarily have high computing power (PDAs, mobile phones, independent sensors...).

The functionalities of the systems based on OBaMADE are implemented as individual services or applications. This is how they can be used in different applications, making small modifications to adapt them to the different situations they could face. The functionalities can also be replicated to obtain a better performance in high demanding scenarios.

The distributed point of view of OBaMADE allows the system to initialize or stop services in an independent way, without affecting the rest of the components of the system. The presented architecture represents an open proposal that can be easily applied to different kind of problems and that can be adapted to cover different needs and knowledge fields. The OBaMADE architecture has successfully been applied to two different case studies, demonstrating the theoretical advantages previously analyzed.

7.4. FUTURE WORK

The investigation presented in this PhD. thesis represents an innovation in the distributed environment field and generates a significant number of future possibilities where this new architecture can be applied and improved. Next, some of the future lines of work are explained.

As outlined before, some crashes were produced in the system when a high number of requests are made at the same time. It is important to reduce the number of crashes, or even to completely eliminate them, to avoid user frustration and bad results in a real life scenario.

The two case studies presented in this document are the current applications made with this architecture. Its validity has been demonstrated,

but it is necessary to apply the prediction model to other knowledge fields to completely check the appropriateness of the architecture in terms of generalization and flexibility.

OBaMADE can be applied to solve different kinds of problems. Currently it has helped to solve prediction generation problems. But, as explained in previous chapters, the architecture proposed in this document can be applied to solve other kinds of problems, such as classification, clustering, planning, etc. It is important to create new applications where the architecture should be slightly modified to be adapted to the new conditions and problems to be solved.

The artificial intelligence techniques applied in the OBaMADE architecture have proved to be useful to solve the proposed problems. But it will be interesting to have new techniques at our disposal (which are constantly appearing) or even more than one possibility for the different steps carried out. Increasing the number of possible solutions will enrich the final solution and the evolution of the architecture.

It is necessary to perform more exhaustive tests to evaluate every single detail of the proposed architecture in terms of time, simplicity and quality of analysis and design. The quality of the results generated by the systems created within the structure of this architecture must also be validated.

" No man is wise enough by himself." *Titus Maccius Plautus*

REFERENCES

Aamodt, A. (1991) A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning. *Knowledge Engineering and Image Processing Group. University of Trondheim.*

Aamodt, A. & Plaza, E. (1994) Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications,* 7**,** 39-59.

Abdallah, S. & Lesser, V. (2004) Organization-based cooperative coalition formation. *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Techonology, IAT.*

Abramson, D., Sosic, R., Giddy, J. & Hall, B. (1995) Nimrod: a tool for performing parametrised simulations using distributed workstations. *Proc. 4th IEEE Symp. on High Performance Distributed Computing.*

Aha, D. W., Molineaux, M. & Ponsen, M. (2005) Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. *Case-Based Reasoning Research and Development.*

Ahuja, M. K. & Carley, K. M. (1999) Network structure in virtual organizations. *Organization Science***,** 741-757.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S. & Foster, I. (2001) Secure, efficient data transport and replica management for high-performance data-intensive computing. *Mass Storage Conference.*

Allende, I. (2003) *El Reino del Dragón de Oro,* Barcelona, Areté.

Allende, I. (2005) *La ciudad de las bestias,* Barcelona, Areté.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D. & Thatte, S. (2003) Business process execution language for web services, version 1.1. *Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation*.

Arditi, D. & Tokdemir, O. B. (1999) Comparison of case-based reasoning and artificial neural networks. *Journal of computing in civil engineering,* 13**,** 162-169.

Argente, E., Julian, V. & Botti, V. (2006) Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science,* 150**,** 55-71.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., Mcinnes, L., Parker, S. & Smolinski, B. (1999) Toward a common component architecture for high-performance scientific computing. *Proc. 8th IEEE Symp. on High Performance Distributed Computing.*

Arrue, B. C., Ollero, A. & Matinez De Dios, J. R. (2000) An intelligent system for false alarm reduction in infrared forest-fire detection. *Intelligent Systems and Their Applications, IEEE* 15**,** 64-73.

Artikis, A. (2003) Executable specification of open norm-governed computational systems. *Department of Electrical & Electronic Engineering.* London, Imperial College

Artikis, A., Kamara, L. & Pitt, J. (2001) Towards an open agent society model and animation. *Proc. of the 2nd. Agent-Based Simulation Workshop.*

Auel, J. M. (2005) *El clan del oso cavernario,* Madrid, El País.

Austin, J. L. (1962) How to Do Things with Words *Harvard University.*

Avery, P. & Foster, I. (2000) The griphyn project: Towards petascale virtual data grids. *The 2000 NSF Information and Technology Research Program*.

Avery, P., Foster, I., Gardner, R., Newman, H. & Szalay, A. (2001) An International Virtual-Data Grid Laboratory for Data Intensive Science. *Technical Report GriPhyN-2001-2.*

Axelrod, R. (1986) An evolutionary approach to norms. *The American Political Science Review***,** 1095-1111.

Ayd, N, U., Murat, S., Olcay Taner, Y., Ld & Ethem, A. (2009) Incremental construction of classifier and discriminant ensembles. *Information Sciences,* 179**,** 1298-1318.

Baker, F. (1995) Requirements for IP version 4 routers. RFC 1812, June 1995.

Barbery, M. (2007) *La elegancia del erizo,* Barcelona, Seix Barral.

Baru, C., Moore, R., Rajasekar, A. & Wan, M. (1998) The SDSC storage resource broker. *Proc. CASCON'98 Conference.* IBM Press.

Baruque, B. & Corchado, E. (2007) Fusion of Visualization Induced SOM. *Innovations in Hybrid Intelligent Systems***,** 151.

Baruque, B., Corchado, E., Mata, A. & Corchado, J. M. (2010) A forecasting solution to the oil spill problem based on a hybrid intelligent system. *Information Sciences,* 180**,** 2029-2043.

Baruque, B., Corchado, E., Rovira, J. & Gonzalez, J. (2008) Application of Topology Preserving Ensembles for Sensory Assessment in the Food Industry. *Intelligent Data Engineering and Automated Learning (IDEAL 2008).*

Baruque, B., Corchado, E. & Yin, H. (2007) ViSOM Ensembles for Visualization and Classification. *International Work Conference on Artificial Neural Networks (IWANN'07), San Sebastián, Spain, Springer, Heidelberg*.

Baruque, B., Corchado, E. S., Mata, A. & Corchado, J. M. (2009) Ensemble Methods for Boosting Visualization Models. *10th International Work-Conference on Artificial Neural Networks (IWANN2009).*

Bayly, J. (2005) *Y de repente, un ángel,* Barcelona, Planeta.

Beavers, G. & Hexmoor, H. (2001) Teams of agents. *Proceedings of the IEEE Systems, Man, and Cybernetics Conference.*

Beegle-Krause, C. J. (1999) GNOME: NOAA's next-generation spill trajectory model. *OCEANS'99 MTS/IEEE. Riding the Crest into the 21st Century,* 3**,** 1262-1266.

Beiriger, J. I., Bivens, H. P., Humphreys, S. L., Johnson, W. R. & Rhea, R. E. (2000) Constructing the ASCI computational grid. *Proc. 9th IEEE Symposium on High Performance Distributed Computing.*

Belore, R. (2005) The SL Ross oil spill fate and behavior model: SLROSM. *Spill Science and Technology Bulletin.*

Benger, W., Foster, I., Novotny, J., Seidel, E., Shalf, J., Smith, W. & Walker, P. (1999) Numerical relativity in a distributed environment. *Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing.*

Benmecheta, A. & Lansari, A. (2007) Monitoring of Oil Pollution by GIS and Remote-Sensing case Of West Algeria Harbours. *Signal Processing and Information Technology, 2007 IEEE International Symposium on***,** 874-879.

Berman, F. (1999) High-performance schedulers. *The Grid: Blueprint for a New Computing Infrastructure***,** 279–309.

Berman, F. D., Wolski, R., Figueira, S., Schopf, J. & Shao, G. (1996) Application-level scheduling on distributed heterogeneous networks. *Conference on High Performance Networking and Computing.* Pittsburgh, Pennsylvania, IEEE Computer Society Washington, DC, USA.

Beynon, M., Ferreira, R., Kurc, T., Sussman, A. & Saltz, J. (2000) DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. *Proc. 8th Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems.*

Bianchini, G. (2006) Wildland Fire Prediction based on Statistical Analysis of Multiple Solutions. *Computer Architecture and Operating Systems Department.* Barcelona, Autonomous University of Barcelona.

Biron, P. V. & Malhotra, A. (2001) XML schema part 2: Datatypes. *W3C recommendation,* 2**,** 2-20010502.

Birrell, A. D. & Nelson, B. J. (1984) Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS),* 2**,** 39-59.

Boggino, A. S. G. (2005) ANEMONA: una metodología multi-agente para sistemas holónicos de fabricación. *PhD Thesis.* Universidad Politécnica de Valencia.

Boissier, O., Hubner, J. F. & Sichman, J. S. (2007) Organization Oriented Programming, from closed to open organizations. *Engineering Societies in the Agents World VI, Sixth International Workshop, ESAW06.* Dublin, Ireland, Springer.

Bond, A. H. & Gasser, L. (1988) An analysis of problems and research in DAI. *in* ALAN H. BOND AND LES GASSER (Ed.) *Readings in Distributed Artificial Intelligence.*

Bongaerts, L. (1998) Integration of scheduling and control in holonic manufacturing systems. *PhD Thesis.* Belgium, Katholieke Universiteit Leuven.

Bratman, M. E. (1987) *Intentions, plans and practical reason,* Cambridge, MA, USA, Harvard University Press.

Bratman, M. E., Israel, D. J. & Pollack, M. E. (1988) Plans and resource-bounded practical reasoning. *Computational intelligence,* 4**,** 349-355.

Breban, S. & Vassileva, J. (2001) Long-term coalitions for the electronic marketplace. *Proceedings of Canadian AI Workshop on Novel E-Commerce Applications of Agents.*

Breiman, L. (1996) Bagging Predictions. *Machine Learning,* 24**,** 123-140.

Brillinger, D. R., Preisler, H. K. & Benoit, J. W. (2003) Risk assessment: a forest fire example. *Statistics and Science: A Festschrift for Terry Speed,* 40**,** 177–196.

Broersen, J., Dastani, M., Hulstijn, J., Huang, Z. & Van Der Torre, L. (2001) The BOID architecture: conflicts between beliefs, obligations, intentions and desires. *Proceedings of the 5th International Conference on Autonomous Agents.* ACM New York, NY, USA.

Brooks, C. H. & Durfee, E. H. (2002) Congregating and market formation. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems.* ACM New York, NY, USA.

Brooks, C. H. & Durfee, E. H. (2003) Congregation formation in multiagent systems. *Autonomous agents and multi-agent systems,* 7**,** 145-170.

Brooks, C. H., Durfee, E. H. & Armstrong, A. (2000) An introduction to congregating in multi-agent systems. *Proceedings of the Fourth International Conference on Multiagent Systems.*

Brooks, R. (1986) A robust layered control system for a mobile robot. *IEEE journal of robotics and automation,* 2**,** 14-23.

Brovchenko, I., Kuschan, A., Maderich, V. & Zheleznyak, M. (2002) The modeling system for simulation of the oil spills in the Black Sea. *3rd EuroGOOS Conference: Building the European capacity in operational oceanography.***,** 192.

Bussmann, S. & Schild, K. (2000) Self-organizing manufacturing control: an industrial application ofagent technology. *Proceedings of the 4th International Con- ference on Multi-Agent Systems (ICMAS 2000).*

Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J. & Kesselman, C. (2000) A national-scale authentication infrastructure. *Computer,* 33**,** 60-66.

Buyya, R. (2002) High Performance Cluster Computing: Architectures and Systems, Volume 1. *Beijing: Publishing House of People's Post and Telecommunication.*

Cai, G., Wu, J., Xue, Y., Wan, W. & Huang, X. (2007) Oil spill detection from thermal anomaly using ASTER data in Yinggehai of Hainan, China. *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International***,** 898-900.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J. & Evans, R. (2002) Agent Oriented Analysis using MESSAGE/UML. *Lecture Notes in Computer Science,* 2222**,** 119-135.

Cardoso, H. L. & Oliveira, E. (2004) Virtual enterprise normative framework within electronic institutions. *Proceedings of the 5th Int. Workshop on Engineering Societies in the Agents World (ESAW 04).* Springer.

Carley, K. M. (1997) Organizational adaptation. *Annals of Operations Research,* 75**,** 25-47.

Carley, K. M. & Gasser, L. (1999) Computational organization theory. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, MA.

Carracedo, P., Torres-López, S., Barreiro, M., Montero, P., Balseiro, C. F., Penabad, E., Leitao, P. C. & Pérez-Munuzuri, V. (2006) Improvement of pollutant drift forecast system applied to the Prestige oil spills in Galicia Coast (NW of Spain): Development of an operational system. *Marine Pollution Bulletin,* 53**,** 350-360.

Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M. & Botti, V. (2007) Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems With Applications,* 34**,** 2-17.

Carriero, N. & Gelernter, D. (1992) Coordination languages and their significance. *Communications of the ACM,* 35**,** 97-107.

Casanova, H. & Dongarra, J. (1997) NetSolve: A network-enabled server for solving computational science problems. *International Journal of High Performance Computing Applications,* 11**,** 212.

Casanova, H., Dongarra, J., Johnson, C. & Miller, M. (1998) Application-specific tools. *in* FOSTER, I. & KESSELMAN, C. (Eds.) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Casanova, H., Obertelli, G., Berman, F. & Wolski, R. (2000) The AppLeS Parameter Sweep Template: User-level middleware for the Grid. *Scientific Programming,* 8**,** 111-126.

Castelfranchi, C. (1990) Social power: A point missed in multi-agent, DAI and HCI. *Decentralized AI***,** 49–62.

Castelfranchi, C. (1998) Modeling social action for AI agents. *Artificial Intelligence,* 103**,** 157-182.

Castelfranchi, C. (2000) Engineering social order. *Lecture Notes in Artificial Intelligence,* 1972**,** 1-18.

Cerami, E. (2002) *Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, O'Reilly & Associates, Inc.

Cervenka, R. & Trencansky, I. (2007) *The Agent Modeling Language-AML: A Comprehensive Approach to Modeling Multi-Agent Systems*, Birkhauser.

Cochrane, M. A. (2002) Se extienden como un reguero de pólvora: Incendios en bosques tropicales en América Latina y el Caribe: Prevención, evaluación y alerta temprana. *Programa de las Naciones Unidas para el Medio Ambiente.* México D.F., United Nations.

Cohen, P. R. & Levesque, H. J. (1990) Intention is choice with commitment. *Artificial Intelligence,* 42**,** 213-261.

Colombetti, M., Fornara, N. & Verdicchio, M. (2004) A social approach to communication in multiagent systems. *Lecture Notes in Artificial Intelligence,* 2990**,** 191-220.

Collins, J., Tsvetovat, M., Mobasher, B. & Gini, M. (1998) MAGNET: A multi-agent contracting system for plan execution. *Proceedings of Workshop on Artificial Intelligence and Manufacturing: State of the Art and State of Practice.* AAAI Press.

Collis, J. C. & Lee, L. C. (1999) Building electronic marketplaces with the ZEUS agent tool-kit. *Lecture Notes in Computer Science,* 1571**,** 1-24.

Copeland, G. & Thiam-Yew, W. (2006) Current data assimilation modelling for oil spill contingency planning. *Environmental Modelling and Software,* 21**,** 142-155.

Corchado, E., Mata, A., Baruque, B., Corchado, J. M. & Pérez-Lancho, B. (2010) An Hybrid Artificial Intelligence System for Forest Fire Forecasting. *International Journal of Computer Mathemathics,* In press.

Corchado, J. M. & Aiken, J. (2002) Hybrid artificial intelligence methods in oceanographic forecasting models. *IEEE SMC Transactions,* 32**,** 307-313.

Corchado, J. M., Bajo, J. & Abraham, A. (2008) GERAmI: Improving the delivery of health care. *IEEE Intelligent Systems. Special Issue on Ambient Intelligence,* 3**,** 19-25.

Corchado, J. M. & Mata, A. (2008) Predicting the Presence of Oil Slicks After an Oil Spill. *Lecture Notes in Artificial Intelligence***,** 573-586.

Corkill, D. D. & Lander, S. E. (1998) Diversity in agent organizations. *Object Magazine,* 8**,** 41-47.

Corkill, D. D. & Lesser, V. R. (1983) The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*

Cotton, D. (2007) MARCOAST-Operational Marine Oil Spill and Water Quality Monitoring Services. *OCEANS 2007-Europe***,** 1-5.

Coulouris, G. F., Dollimore, J. & Kindberg, T. (2005) *Distributed systems: concepts and design*, Addison-Wesley Longman.

Cuni, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C. & Solchaga, T. (2004) MASFIT: Multi-agent system for fish trading. *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI'2004).* IOS Press.

Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T. R., Mark, W. S., Tenenbaum, J. M. & Weber, J. C. (1993) PACT: An experiment in integrating concurrent engineering systems. *Computer,* 26**,** 28-37.

Czajkowski, K., Fitzgerald, S., Foster, I. & Kesselman, C. (2001) Grid information services for distributed resource sharing. *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing.* San Francisco.

Chandrasekaran, B. (1981) Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man and Cybernetics,* 11**,** 1-5.

Chang, C. L. (2005) Using case-based reasoning to diagnostic screening of children with developmental delay. *Expert Systems With Applications,* 28**,** 237-247.

Channabasavaiah, K., Holley, K. & Tuggle, E. (2003) Migrating to a service-oriented architecture. *IBM DeveloperWorks,* 16.

Chavez, A. & Maes, P. (1996) Kasbah: An agent marketplace for buying and selling goods. *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96).* London, UK.

Chen, D. & Burrell, P. (2001) Case-based reasoning system and artificial neural networks: a review. *Neural Computing & Applications,* 10**,** 264-276.

Chen, F., Wang, C., Zhang, H., Zhang, B. & Wu, F. (2007) SAR images classification using case-based reasoning method. *Geoscience and Remote Sensing Symposium, IGARSS 2007.*

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. & Tuecke, S. (2000) The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications,* 23**,** 187-200.

Childers, L., Disz, T., Olson, R., Papka, M. E., Stevens, R. & Udeshi, T. (2000) Access grid: Immersive group-to-group collaborative visualization. *Proc. 4th International Immersive Projection Technology Workshop.*

Chow, H. K. H., Choy, K. L., Lee, W. B. & Lau, K. C. (2006) Design of a RFID case-based resource management system for warehouse operations. *Expert Systems With Applications,* 30**,** 561-576.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001) Web services description language (WSDL) 1.1.

Chua, D. K. H., Li, D. Z. & Chan, W. T. (2001) Case-based reasoning approach in bid decision making. *Journal of construction engineering and management,* 127**,** 35.

Chun, S. H. & Park, Y. J. (2005) Dynamic adaptive ensemble case-based reasoning: application to stock market prediction. *Expert Systems With Applications,* 28**,** 435-443.

Chvatal, V. (1979) A greedy heuristic for the set-covering problem. *Mathematics of operations research***,** 233-235.

Damoah, R., Spichtinger, N., Forster, C., James, P., Mattis, I., Wandinger, U., Beirle, S., Wagner, T. & Stohl, A. (2004) Around the world in 17 days–hemispheric-scale transport of forest fire smoke from Russia in May 2003. *Atmospheric Chemistry and Physics,* 4**,** 1311–1321.

Dautenhahn, K. (2000) Reverse engineering of societies: a biological perspective. *Proc. of AISB Symposium "Starting from Society - application of social analogies to computational systems".* University of Birmingham, England, AISB.

Davidsson, P. (2001) Categories of artificial societies. *Engineering Societies in the Agents World II, LNAI 2203***,** 1-9.

Davis, R. & Smith, R. G. (1980) Negotiation as a metaphor for distributed problem solving. *Communication in Multiagent Systems. Agent Communication Languages and Conversation Policies***,** 51–97.

De Mántaras, R. L. & Plaza, E. (1997) Case-Based Reasoning: An Overview. *AI Communications,* 10**,** 21-29.

Decker, K. (1996) TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. *in* JENNINGS, G. O. H. A. N. (Ed.) *Foundations of Distributed Artificial Intelligence.* Wiley Inter-Science.

Decker, K., Lesser, V., Prasad, M. V. N. & Wagner, T. (1995) MACRON: an architecture for multi-agent cooperative information gathering. *Proccedings of the CIKM Workshop on Intelligent Information Agents.*

Decker, K. & Li, J. (1998) Coordinated hospital patient scheduling. *In Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98).* IEEE Computer Society.

Decker, K., Sycara, K. & Williamson, M. (1997) *Middle-agents for the internet,* Lawrence Erlbaum Associates Ltd. .

Decker, K. S. & Lesser, V. R. (1992) Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems,* 1**,** 319-346.

Decker, K. S. & Lesser, V. R. (1993) Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management,* 2**,** 215-234.

Defanti, T. & Stevens, R. (1999) Teleimmersion. *in* FOSTER, I. & KESSELMAN, C. (Eds.) *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann.

Delany, S. J. (2006) Using Case-Based Reasoning for Spam Filtering. *Dublin Institute of Technology.*

Dellarocas, C. & Klein, M. (2000a) Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. *Lecture Notes in Computer Science,* 1788**,** 24-39.

Dellarocas, C. & Klein, M. (2000b) Contractual Agent Societies: Negotiated shared context and social control in open multi-agent systems. *Social Order in Multi-Agent Systems***,** 113-133.

Den Breejen, E., Breuers, M., Cremer, F., Kemp, R., Roos, M., Schutte, K. & De Vries, J. S. (1998) Autonomous Forest Fire Detection. *III International Conference on Forest Fire Research.*

Dennet, D. C. (1987) *The intentional stance,* Cambridge, MA, MIT Press.

Dialani, V. (2002) UDDI-M Version 1.0 API Specification. *University of Southampton–UK,* 2.

Diaz, F., Fdez-Riverola, F. & Corchado, J. M. (2006) Gene-CBR: A case-based reasoning tool for cancer diagnosis using microarray data sets. *Computational Intelligence,* 22**,** 254-268.

Dignum, F., Morley, D., Sonenberg, E. A. & Cavedon, L. (2000) Towards socially sophisticated BDI agents. *Proceedings of the 4th International Conference on Multi-Agent Systems.* Boston.

Dignum, V. (2004) A Model for Organizational Interaction: based on Agents, founded in Logic. *PhD thesis.* Utrecht University.

Dignum, V. & Dignum, F. (2001) Modelling agent societies: coordination frameworks and institutions. *Progress in Artificial Intelligence, LNAI,* 2258**,** 191-204.

Dignum, V. & Dignum, F. (2007a) A landscape of agent systems for the real world. *Technical Report* Utrecht University.

Dignum, V. & Dignum, F. (2007b) A logic for agent organizations. *FAMAS@ Agents***,** 3-7.

Dignum, V., Meyer, J. J., Weigand, H. & Dignum, F. (2002) An organization-oriented model for agent societies. *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications*

Dignum, V., Weigand, H. & Xu, L. (2002) Agent Societies: Towards framework-based design *in* WOOLDRIDGE, M., WEISS, G. & P., C. (Eds.) *Agent-Oriented Software Engineering II*. LNCS, Springer-Verlag.

Douligeris, C., Collins, J., Iakovou, E., Sun, P., Riggs, R. & Mooers, C. N. K. (1995) Development of OSIMS: An oil spill information management system. *Spill Science & Technology Bulletin*, 2**,** 255-263.

Dunin-Keplicz, B. & Verbrugge, R. (2002) Collective intentions. *Fundamenta Informaticae*, 51**,** 271-295.

Durfee, E. H. & Lesser, V. R. (1991) Partial global planning: a coordination framework for distributedhypothesis formation. *IEEE Transactions on Systems, Man and Cybernetics*, 21**,** 1167-1183.

Durfee, E. H., Lesser, V. R. & Corkill, D. D. (1987) Coherent cooperation among communicating problem solvers. *IEEE Transactions on computers*, 36**,** 1275-1291.

Durfee, E. H., Lesser, V. R. & Corkill, D. D. (1989) Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering*, 1**,** 63-83.

Edmonds, B. (1999) Capturing Social Embeddedness: a constructivist approach. *Adaptive Behavior*, 7**,** 323-348.

Elhakeem, A. A., Elshorbagy, W. & Chebbi, R. (2007) Oil Spill Simulation and Validation in the Arabian (Persian) Gulf with Special Reference to the UAE Coast. *Water, Air, & Soil Pollution*, 184**,** 243-254.

Erickson, J. & Siau, K. (2008) Web Services, Service-Oriented Computing, and Service-Oriented Architecture: Separating Hype from Reality. *Journal of Database Management*, 19**,** 42-54.

Esteva, M., De La Cruz, D. & Sierra, C. (2002) ISLANDER: an electronic institutions editor. *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*. ACM New York, NY, USA.

Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Garcia, P. & Arcos, J. L. (2001) On the formal specifications of electronic institutions. *Lecture Notes in Computer Science***,** 126-147.

Etzioni, O. & Weld, D. (1994) A softbot-based interface to the internet. *Communications of the ACM,* 37**,** 72-76.

Excelente-Toledo, C. B. & Jennings, N. R. (2004) The dynamic selection of coordination mechanisms. *Autonomous agents and multi-agent systems,* 9**,** 55-85.

Fatima, S. S. & Wooldridge, M. (2001) Adaptive task resources allocation in multi-agent systems. *Proceedings of the Fifth International Conference on Autonomous Agents.* Montreal, Canada, ACM New York, NY, USA.

Fdez-Riverola, F. & Corchado, J. M. (2004) FSfRT: Forecasting System for Red Tides. *Applied Intelligence,* 21**,** 251-264.

Fdez-Riverola, F., Díaz, F. & Corchado, J. M. (2007) Reducing the Memory Size of a Fuzzy Case-Based Reasoning System Applying Rough Set Techniques. *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* 37**,** 138-146.

Fdez-Riverola, F., Iglesias, E. L., Díaz, F., Méndez, J. R. & Corchado, J. M. (2007a) Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems With Applications,* 33**,** 36-48.

Fdez-Riverola, F., Iglesias, E. L., Díaz, F., Méndez, J. R. & Corchado, J. M. (2007b) SpamHunting: An instance-based reasoning system for spam labelling and filtering. *Decision Support Systems,* 43**,** 722-736.

Ferber, J. & Gutknecht, O. (1998) A meta-model for the analysis and design of organizations in multi-agent systems. *Proc. of the 3rd. International Conference on Multi-Agent Systems.* IEEE Computer Society.

Ferber, J., Gutknecht, O. & Michel, F. (2004) From agents to organizations: an organizational view of multi-agent systems. *Lecture Notes in Computer Science***,** 214-230.

Ferguson, I. A. (1995) Integrated control and coordinated behaviour: A case for agent models. *Lecture Notes in Artificial Intelligence,* 890**,** 203-203.

Finin, T., Fritzson, R., Mckay, D. & Mcentire, R. (1994) KQML as an agent communication language. *Proceedings of the third international conference on Information and knowledge management.* Gaithersburg, Maryland, United States, ACM New York, NY, USA.

Fipa, T. C. (2002) Communication, FIPA Communicative Act Library Specification, SC00037J.

Fischer, K. (1999) Agent-based design of holonic manufacturing systems. *Robotics and autonomous Systems,* 27**,** 3-13.

Fisher, M. (1994) A Survey of Concurrent METATEM-the Language and its Applications. *Lecture Notes in Computer Science,* 827**,** 480-480.

Fitoussi, D. & Tennenholtz, M. (2000) Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence,* 119**,** 61-102.

Fitzgerald, S., Foster, I., Kesselman, C., Von Laszewski, G., Smith, W. & Tuecke, S. (1997) A directory service for configuring high-performance distributed computations. *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing.*

Flores, F. & Ludlow, J. J. (1976) Doing and Speaking in the Office. *Decision Support Systems: Issues and Challenges***,** 95-118.

Follett, K. (2007) *Un mundo sin fin,* Barcelona, Plaza y Janés.

Foster, I., Jennings, N. R. & Kesselman, C. (2004) Brain meets brawn: Why grid and agents need each other. *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004).* IEEE Computer Society.

Foster, I. & Karonis, N. T. (1998) A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. *Proc. SC'98.* IEEE Computer Society Washington, DC, USA.

Foster, I. & Kesselman, C. (1999) *The grid: blueprint for a future computing infrastructure,* Morgan Kaufmann Publishers USA.

Foster, I., Kesselman, C., Tsudik, G. & Tuecke, S. (1998) A security architecture for computational grids. *ACM Conference on Computers and Security.* ACM New York, NY, USA.

Foster, I., Vockler, J., Wilde, M. & Zhao, Y. (2002) Chimera: A virtual data system for representing, querying, and automating data derivation. *Proceedings of the 14th International Conference on Scientific and Statistical Database Management.* Edinburgh.

Fox, M. S. (1979) *Organization structuring: Designing large complex software,* Carnegie-Mellon University, Dept. of Computer Science.

Fox, M. S. (1981) An organizational view of distributed systems. *IEEE Transactions on Systems, Man and Cybernetics,* 11**,** 70-80.

Fox, M. S., Barbuceanu, M., Gruninger, M. & Lin, J. (1998) An organization ontology for enterprise modelling. *Simulating organizations: Computational models of institutions and groups***,** 131–152.

Fox, M. S. & Gruninger, M. (1998) Enterprise modeling. *AI magazine,* 19**,** 109-121.

Franklin, S. & Graesser, A. (1997) Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. *Lecture Notes in Computer Science,* 1193**,** 21-36.

Gabriel, E., Resch, M., Beisel, T. & Keller, R. (1998) Distributed computing in a heterogeneous computing environment. *Proc. EuroPVM/MPI'98.*

Galbraith, J. R. (1974) Organization design: An information processing view. *Interfaces***,** 28-36.

Gale, W. A. (2009) Statistical applications of artificial intelligence and knowledge engineering. *The Knowledge Engineering Review,* 2**,** 227-247.

Galushka, M. & Patterson, D. (2006) Intelligent index selection for case-based reasoning. *Knowledge-Based Systems,* 19**,** 625-638.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995) *Design patterns: elements of reusable object-oriented software,* Reading, MA, Addison-Wesley.

Gannon, D. & Grimshaw, A. (1998) Object-based approaches. *in* FOSTER, I. & KESSELMAN, C. (Eds.) *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann.

Garrido, L. & Sycara, K. (1995) Multi-agent meeting scheduling: Preliminary experimental results. *International Conference on Multi-Agent Systems (ICMAS'95).*

Gasser, L. (1991) Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence,* 47**,** 107-138.

Gasser, L. (2001) Perspectives on organizations in multi-agent systems. *in* LUCK, M., MARIK, V., STEPANKOVA, O. & TRAPPL, R. (Eds.) *Multi-agent Systems and Applications. 9th ECCAI Advanced Course, EASSS 2001.* Springer.

Gateau, B., Boissier, O., Khadraoui, D. & Dubois, E. (2005) MOISE-Inst: An organizational model for specifying rights and duties of autonomous agents. *1st International Workshop on Coordination and Organisation.*

Genesereth, M. R. (1997) An agent-based framework for interoperability. *in* BRADSHAW, J. (Ed.) *Software agents.* MIT Press.

Genesereth, M. R. & Ketchpel, S. P. (1994) Software agents. *Communications of the ACM* 37**,** 48-53, 147.

Genesereth, M. R. & Nilsson, N. J. (1987) *Logical foundations of artificial intelligence,* Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Georgeff, M. & Rao, A. (1998) Rational software agents: from theory to practice. *in* JENNINGS, N. R. & WOOLDRIDGE, M. J. (Eds.) *Agent Technology: Foundations, Applications, and Markets.* Secaucus, NJ, Springer-Verlag New York.

Georgeff, M. P. & Lansky, A. L. (1987) Reactive reasoning and planning. *Proceedings of the 6th National Conference in AI* Seattle, WA.

Gestosa (2005) http ://www.adai.pt/ceif/Gestosa/. *ADAI-CEIF(Center of Forest Fire Studies).*

Gibbins, N., Harris, S. & Shadbolt, N. (2004) Agent-based semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web,* 1**,** 141-154.

Giglio, L., Descloitres, J., Justice, C. O. & Kaufman, Y. J. (2003) An Enhanced Contextual Fire Detection Algorithm for MODIS. *Remote Sensing of Environment,* 87**,** 273-282.

Giret B., A. (2005) Anemona: Una metodología multi-agente para sistema holónicos de fabricación. *Departamento de Sistemas Informáticos y, Computación.* Universidad Politécnica de Valencia.

Gironella, J. M. (1952) *Los cipreses creen en Dios; novela,* Barcelona, Editorial Planeta.

Glass, A. & Grosz, B. J. (2003) Socially conscious decision-making. *Autonomous agents and multi-agent systems,* 6**,** 317-339.

Gnanasambandam, N., Lee, S., Gautam, N., Kumara, S. R. T., Peng, W., Manikonda, V., Brinn, M. & Greaves, M. (2004) Reliable mas performance prediction using queueing models. *Proceedings of the IEEE Multi-agent Security and Survivability Symposium (MASS).*

Gokhale, A. & Schmidt, D. C. (1996) The performance of the CORBA dynamic invocation interface and dynamic skeleton interface over high-speed ATM networks. *Proceedings of GLOBECOM'96.* London, England.

Goux, J. P., Kulkarni, S., Linderoth, J. & Yoder, M. (2000) An enabling framework for master-worker applications on the Computational Grid. *Proc. 9th IEEE Symp. on High Performance Distributed Computing.* IEEE Press.

Griffiths, N. (2003) Supporting cooperation through clans. *Intelligence, Challenges and Advances – Proceedings of the 2nd IEEE Systems, Man and Cybernetics.*

Grinshaw, A. S. & Wm, A. (1996) Wulf and the whole Legion Team. Legion—A view From 50000 Feet. *Proc. 5th IEEE Symposium on High Performance Distributed Computing,.* IEEE Press.

Grosz, B. J. & Kraus, S. (1996) Collaborative plans for complex group action. *Artificial Intelligence,* 86**,** 269-357.

Grosz, B. J. & Sidner, C. L. (1990) Plans for discourse. *Intentions in communication***,** 417-444.

Gupta, U. G. (1994) How case-based reasoning solves new problems. *Interfaces***,** 110-119.

Guttman, R. H., Moukas, A. G. & Maes, P. (2001) Agent-mediated electronic commerce: A survey. *The Knowledge Engineering Review,* 13**,** 147-159.

Haddadi, A. (1996) *Communication and cooperation in agent systems: a pragmatic theory*, Springer.

Harkavy, M. (1996) Webster's new encyclopedic dictionary. *Black Dog & Leventhal publishers Inc,* 151.

Haupt, S. E., Pasini, A. & Marzban, C. (2008) *Artificial Intelligence Methods in the Environmental Sciences*, Springer Publishing Company, Incorporated.

Hayden, S. C., Carrick, C. & Yang, Q. (1999) A catalog of agent coordination patterns. *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents.* ACM New York, NY, USA.

Hendler, J. (2001) Agents and the semantic web. *IEEE Intelligent Systems,* 16**,** 30-37.

Heskes, T. (1997) Balancing between bagging and bumping. *Advances in Neural Information Processing Systems,* 9**,** 466-472.

Hewitt, C. (1986) Offices are open systems. *ACM Transactions on Information Systems (TOIS),* 4**,** 271-287.

Hexmoor, H. & Beavers, G. (2001) Towards teams of agents. *Proceedings of the International Conference in Artificial Intelligence (IC-AI'2001).* CSREA Press.

Hodge, B. J., Anthony, W. P., Gales, L. M. & Ruiz, D. (1998) *Teoría de la organización: Un enfoque estratégico*, Prentice Hall.

Horling, B. (2003) Using autonomy, organizational design and negotiation in a distributed sensor network. *Distributed Sensor Networks: A multiagent perspective***,** 139-183.

Horling, B., Benyo, B. & Lesser, V. (2001) Using self-diagnosis to adapt organizational structures. *Proceedings of the 5th International Conference on Autonomous Agents* ACM New York, NY, USA.

Horling, B. & Lesser, V. (2005) Analyzing, modeling and predicting organizational effects in a distributed sensor network. *Journal of the*

Brazilian Computer Society, Special Issue on Agents Organizations, 11**,** 9-30.

Horling, B., Mailler, R. & Lesser, V. (2004) A case study of organizational effects in a distributed sensor network. *Computer Science Technical Report,* 04-03.

Horling, B., Mailler, R., Shen, J., Vincent, R. & Lesser, V. (2003) Using autonomy, organizational design and negotiation in a distributed sensor network. *Distributed Sensor Networks: A multiagent perspective,* 139-183.

Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H. & Stockinger, K. (2000) Data management in an international data grid project. *Proc. 1st IEEE/ACM International Workshop on Grid Computing.*

Howell, J., Design, C. & Kotz, D. (2000) End-to-end authorization. *Proc. Symposium on Operating Systems Design and Implementation.* USENIX Association.

Huang, M. J., Chen, M. Y. & Lee, S. C. (2007) Integrating data mining with case-based reasoning for chronic diseases prognosis and diagnosis. *Expert Systems With Applications,* 32**,** 856-867.

Hubner, J. F., Sichman, J. S. & Boissier, O. (2002) A model for the structural, functional, and deontic specification of organizations in multiagent systems. *Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA'02).*

Hubner, J. F., Sichman, J. S. & Boissier, O. (2005) S-Moise: A Middleware for developing Organised Multi-Agent Systems. *In Proc. Int. Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS,*. Springer.

Huhns, M. N. & Singh, M. P. (1998) Agents and multiagent systems: Themes, approaches, and challenges. *Readings in agents,* 1-23.

Huhns, M. N. & Stephens, L. M. (1999) Multiagent Systems and Societies of Agents. *in* WEISS, G. (Ed.) *Multi-agent Systems: a Modern Approach to Distributed Artificial Intelligence.* MIT Press.

Iliadis, L. S. (2005) A decision support system applying an integrated fuzzy model for long-term forest fire risk estimation. *Environmental Modelling and Software,* 20**,** 613-621.

Im, K. H. & Park, S. C. (2007) Case-based reasoning and neural network based expert system for personalization. *Expert Systems With Applications,* 32**,** 77-85.

Ishida, T., Gasser, L. & Yokoo, M. (1992) Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*.

Jennings, N. & Wooldridge, M. (1996) Software agents. *IEEE review*, 42**,** 17-20.

Jennings, N. R. (1995) Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75**,** 195-240.

Jennings, N. R. (1999) *Agent-based computing: Promise and perils*, Lawrence Erlbaum Associates Ltd.

Jennings, N. R., Sycara, K. & Wooldridge, M. (1998) A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems Journal*, 1**,** 7-38.

Jennings, N. R. & Wooldridge, M. (1995) Applying agent technology. *Applied Artificial Intelligence*, 9**,** 351-361.

Jennings, N. R. & Wooldridge, M. (1999) Agent-oriented software engineering. *Lecture Notes in Computer Science***,** 4-10.

Jennings, N. R. & Wooldridge, M. J. (1998) *Agent technology: foundations, applications, and markets*, Springer-Verlag New York, Inc. Secaucus, NJ, USA.

Jensen, D. & Lesser, V. (2002) Social pathologies of adaptive agents. *Safe Learning Agents: Papers from the 2002 AAAI Spring Symposium*. AAAI Press.

Jha, M. N., Levy, J. & Gao, Y. (2008) Advances in Remote Sensing for Oil Spill Disaster Management: State-of-the-Art Sensors Technology for Oil Spill Surveillance. *Sensors*, 8**,** 236-255.

Jian-Dong, G. U. O. & Shang-Liang, Z. (2007) Multi threading implementations in real-time CORBA [J]. *Computer Engineering and Design*, 2.

Jonker, C. M., Klusch, M. & Treur, J. (2000) Design of collaborative information agents. *Lecture Notes in Computer Science***,** 262-284.

Jorda, G., Comerma, E., Bolanos, R. & Espino, M. (2007) Impact of forcing errors in the CAMCAT oil spill forecasting system. A sensitivity study. *Journal of Marine Systems*, 65**,** 134-157.

Jordi, A., Ferrer, M. I., Vizoso, G., Orfila, A., Basterretxea, G., Casas, B., Álvarez, A., Roig, D., Garau, B. & Martínez, M. (2006) Scientific management of Mediterranean coastal zone: A hybrid ocean forecasting system for oil spill and search and rescue operations. *Marine Pollution Bulletin*, 53**,** 361-368.

Juan, T., Pearce, A. & Sterling, L. (2002) ROADMAP: Extending the Gaia Methodology for Complex Open Systems. *Proceedings of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems.* ACM Press.

Jung, H., Tambe, M. & Kulkarni, S. (2001) Argumentation as distributed constraint satisfaction: Applications and results. *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents.* ACM New York, NY, USA.

Kaminka, G. A., Pynadath, D. V. & Tambe, M. (2002) Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research,* 17**,** 83-135.

Karafyllidis, I. & Thanailakis, A. (1997) A model for predicting forest fire spreading using cellular automata. *Ecological Modelling,* 99**,** 87-97.

Kautz, H., Selman, B., Coen, M., Ketchpel, S. & Ramming, C. (1994) *An experiment in the design of software agents,* John Wiley & sons Ltd. .

Keramitsoglou, I., Cartalis, C. & Kiranoudis, C. T. (2006) Automatic identification of oil spills on satellite images. *Environmental Modelling and Software,* 21**,** 640-652.

Khedro, T. & Genesereth, M. R. (1995) Facilitators: A networked computing infrastructure for distributed software interoperation. *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence in Distributed Information Networks.*

Klein, M., Rodriguez-Aguilar, J. A. & Dellarocas, C. (2003) Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous agents and multi-agent systems,* 7**,** 179-189.

Kleinrock, L. (1994) *Realizing the information future: the Internet and beyond,* National Academy Press.

Klusch, M. (1999) *Intelligent information agents: agent-based information discovery and management on the Internet,* Springer-Verlag New York, Inc. Secaucus, NJ, USA.

Klusch, M. & Gerber, A. (2002) Dynamic coalition formation among rational agents. *IEEE Intelligent Systems,* 17**,** 42-47.

Koestler, A. (1968) The ghost in the machine. *Psychiatric communications,* 10**,** 45.

Kohonen, T. (1995) Self-Organizing Maps. *Springer Series in Information Sciences.* Berlin, Germany, Springer.

Kolodner, J. L. (1991) Improving human decision making through case-based decision aiding. *AI Magazine,* 12**,** 52-68.

Kolodner, J. L. (1993) *Case-based Reasoning*, Morgan Kaufmann.

Kolp, M., Giorgini, P. & Mylopoulos, J. (2003) Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems,* 13**,** 3-25.

Krafzig, D., Banke, K. & Slama, D. (2004) *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*, Prentice Hall PTR Upper Saddle River, NJ, USA.

Kurbel, K. & Loutchko, I. (2003) Towards multi-agent electronic marketplaces: what is there and what is missing? *The Knowledge Engineering Review,* 18**,** 33-46.

Lai, H. (2007) A Service-Oriented Architecture Based Platform to Integrate Information System for Semiconductor Manufacturing. *Extension Education Master Program of Information & Electrical Engineering.* Fench Chia University.

Larson, K. S. & Sandholm, T. W. (2000) Anytime coalition structure generation: An average case study. *Journal of Experimental & Theoretical Artificial Intelligence,* 12**,** 23-42.

Larsson, S. (2008) *La chica que soñaba con una cerilla y un bidón de gasolina*, Barcelona, Ediciones Destino.

Larsson, S. (2009a) *La reina en el palacio de las corrrientes de aire*, Barcelona, Destino.

Larsson, S. (2009b) *Los hombres que no amaban a las mujeres*, Barcelona, Ediciones Destino.

Lashkari, Y., Metral, M. & Maes, P. (1997) Collaborative interface agents. *Readings in agents***,** 111–116.

Leake, D. B. (1996) *Case-based reasoning: Experiences, lessons and future directions*, MIT Press Cambridge, MA, USA.

Lee, J., Yang, J. & Chung, J. Y. (2002) Winslow: A Business Process Management System with Web Services. *Technical Paper*. Electronic Commerce Research Center, National Sun Yat-sen University.

Leigh, J., Johnson, A. & Defanti, T. A. (1997) CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality: Research, Development and Applications,* 2**,** 217-237.

Lerman, K. & Galstyan, A. (2001) A general methodology for mathematical analysis of multi-agent systems. *USC Information Sciences Technical Report ISI-TR-529*.

Lerman, K. & Shehory, O. (2000) Coalition Formation for Large-Scale Electronic Markets. *Int. Conference on Multi-Agent Systems.*

Lesperance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R. & Scherl, R. B. (1996) Foundations of a logical approach to agent programming. *Intelligent Agents II Agent Theories, Architectures, and Languages.* Springer.

Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M. N. & Raja, A. (2004) Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous agents and multi-agent systems,* 9**,** 87-143.

Lesser, V. R. (1991) A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems Man and Cybernetics,* 21**,** 1347-1362.

Lesser, V. R. (1998) Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous agents and multi-agent systems,* 1**,** 89-111.

Lesser, V. R. & Corkill, D. D. (1981) Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man and Cybernetics,* 11**,** 81-96.

Lesser, V. R. & Corkill, D. D. (1983) The distributed vehicle monitoring testbed. *AI magazine,* 4**,** 63-109.

Lesser, V. R. & Erman, L. D. (1980) Distributed interpretation: A model and experiment. *IEEE Transactions on computers,* 100**,** 1144-1163.

Levesque, H. J., Cohen, P. R. & Nunes, J. H. T. (1990) On acting together. *Proceedings of the Eighth National Conference on Artificial Intelligence.* Boston, MA.

Li, H. (1996) Selecting KBES development techniques for applications in the construction industry. *Construction Management and Economics,* 14**,** 67-74.

Li, H., Hu, D., Hao, T., Wenyin, L. & Chen, X. (2007) Adaptation Rule Learning for Case-Based Reasoning. *Third International Conference on Semantics, Knowledge and Grid***,** 44-49.

Li, J. Y., Ni, Z. W., Liu, X. & Liu, H. T. (2006) Case-Base Maintenance Based on Multi-Layer Alternative-Covering Algorithm. *Machine Learning and Cybernetics, 2006 International Conference on***,** 2035-2039.

Li, X. & Yeh, A. G. (2004) Multitemporal SAR images for monitoring cultivation systems using case-based reasoning. *Remote Sensing of Environment,* 90**,** 524-534.

Lin, Z. & Carley, K. M. (1995) DYCORP: A computational framework for examining organizational performance under dynamic conditions. *The Journal of mathematical sociology,* 20**,** 193-217.

Liu, X. & Wirtz, K. W. (2005) Sequential negotiation in multiagent systems for oil spill response decision-making. *Marine Pollution Bulletin,* 50**,** 469-74.

Liu, X. & Wirtz, K. W. (2007) Decision making of oil spill contingency options with fuzzy comprehensive evaluation. *Water Resources Management,* 21**,** 663-676.

Long, D. G. (2001) Mapping fire regimes across time and space: Understanding coarse and fine-scale fire patterns. *International Journal of Wildland Fire,* 10**,** 329-342.

López, F. L. Y., Luck, M. & D'inverno, M. (2006) A normative framework for agent-based systems. *Computational & Mathematical Organization Theory,* 12**,** 227-250.

López, I., Follen, G., Gutiérrez, R., Foster, I., Ginsburg, B. & Larsson, O. (2000) S. Martin and Tuecke, S., NPSS on NASA's IPG: Using CORBA and Globus to Coordinate Multidisciplinary Aeroscience Applications. *NASA HPCC/CAS Workshop.* NASA Ames Research Center.

Luger, G. F. (2002) *Artificial intelligence: structures and strategies for complex problem solving,* Addison Wesley Publishing Company.

Lybäck, D. (1999) Transient diversity in multi-agent systems. *Department of Computer and Systems Sciences.* Stockholm University and the Royal Institute of Technology.

Maes, P. (1990) *Designing autonomous agents: theory and practice from biology to engineering and back,* MIT press.

Maes, P. (1994) Agents that reduce work and information overload. *Communications of the ACM,* 37**,** 30-40.

Maes, P. (1997) Intelligent software. *Proceedings of the 2nd international conference on Intelligent user interfaces.* Orlando, Florida, United States, ACM New York, NY, USA.

Mahmoud, Q. H. (2005) Service-oriented architecture (SOA) and web services: The road to Enterprise Application Integration (EAI). *Technical article, Sun Developer Network.*

Mailler, R. & Lesser, V. (2004) Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems* IEEE Computer Society Washington, DC, USA.

Mailler, R., Lesser, V. & Horling, B. (2003) Cooperative negotiation for soft real-time distributed resource allocation. *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003).* Melbourne, ACM New York, NY, USA.

Malone, T. W. & Crowston, K. (1994) The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR),* 26**,** 87-119.

Malone, T. W. & Smith, S. A. (1988) Modeling the performance of organizational structures. *Operations Research***,** 421-436.

Malzieu, M. (2009) *La mecánica del corazón,* Barcelona, Random House Mondadori.

March, J. G., Simon, H. A. & Guetzkow, H. S. (1958) *Organizations,* John Wiley & Sons Inc.

Marsella, S., Tambe, M., Adibi, J., Al-Onaizan, Y., Kaminka, G. A. & Muslea, I. (2001) Experiences acquired in the design of RoboCup teams: A comparison of two fielded teams. *Autonomous agents and multi-agent systems,* 4**,** 115-129.

Martins Fernandes, P. A. (2001) Fire spread prediction in shrub fuels in Portugal. *Forest Ecology and Management,* 144**,** 67-74.

Massie, J. L. (1973) *Bases esenciales de la Administración,* México D.F., Edicorial Diana.

Mata, A. & Corchado, J. M. (2009) Forecasting the probability of finding oil slicks using a CBR system. *Expert Systems With Applications,* 36**,** 8239-8246.

Mata, A., Pérez-Lancho, B., Gónzalez, A., Baruque, B. & Corchado, E. S. (2009) MACSDE: Multi-Agent Contingency Response System for Dynamic Environments. *HAIS 2009.*

Mathieu, P., Routier, J. C. & Secq, Y. (2002) Dynamic organization of multi-agent systems. *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems.* ACM New York, NY, USA.

Matson, E., Deloach, S. & Kansas State Univ, M. (2003) Using dynamic capability evaluation to organize a team of cooperative, autonomous robots. *Proceedings of The 2003 International Conference on Artificial Intelligence (IC-AI'03).*

Maturana, F., Shen, W. & Norrie, D. H. (1999) MetaMorph: an adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research,* 37**,** 2159-2173.

Mazzeo, G., Marchese, F., Filizzola, C., Pergola, N. & Tramutoli, V. (2007) A Multi-temporal Robust Satellite Technique (RST) for Forest Fire Detection. *Analysis of Multi-temporal Remote Sensing Images, 2007.*

Mccourt, F. (2006) *El profesor,* Madrid, Maeva.

Mcilraith, S. A. & Zeng, T. C. H. (2001) Semantic web services. *IEEE Intelligent Systems,* 16**,** 46-53.

Menemenlis, D., Hill, C., Adcroft, A., Campin, J. M., Cheng, B., Ciotti, B., Fukumori, I., Heimbach, P., Henze, C. & Köhl, A. (2005) NASA Supercomputer Improves Prospects for Ocean Climate Research. *EOS Transactions,* 86**,** 89-95.

Mercier, G. & Girard-Ardhuin, F. (2006) Partially Supervised Oil-Slick Detection by SAR Imagery Using Kernel Expansion. *Geoscience and Remote Sensing, IEEE Transactions on,* 44**,** 2839-2846.

Mérida-Campos, C. & Willmott, S. (2004) Modelling coalition formation over time for iterative coalition games. *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004).* IEEE Computer Society Washington, DC, USA.

Mitra, N. (2003) Soap version 1.2 part 0: Primer. *W3C recommendation,* 24.

Montali, A., Giacinto, G., Migliaccio, M. & Gambardella, A. (2006) Supervised pattern classification techniques for oil spill classification in SAR images: preliminary results. *Proceedings of the SeaSAR,* 2006.

Montani, S. (2007) Case-based Reasoning for managing non-compliance with clinical guidelines. *International Conference on Case-Based Reasoning, ICCBR 2007, Proceedings.*

Montani, S. & Anglano, C. (2008) Achieving self-healing in service delivery software systems by means of case-based reasoning. *Applied Intelligence,* 28**,** 139-152.

Montenegro, R., Plaza, A., Ferragut, L. & Asensio, M. I. (1997) Application of a nonlinear evolution model to fire propagation. *Nonlinear Analysis,* 30**,** 2873-2882.

Montgomery, T. A. & Durfee, E. H. (1993) Search reduction in hierarchical distributed problem solving. *Group Decision and Negotiation,* 2**,** 301-317.

Moreno, R. A., Do Santos, M., Bertozzo, N., De Sa Rebelo, M., Furuie, S. S. & Gutierrez, M. A. (2008) Medical Image distribution and visualization in a hospital using CORBA. *Engineering in Medicine*

and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE Vancouver, Canada.

Moses, Y. & Tennenholtz, M. (1995) Artificial social systems. *Computers and Artificial Intelligence,* 14**,** 533-562.

Moulin, B. & Chaib-Draa, B. (1996) An overview of distributed artificial intelligence. *Foundations of distributed artificial intelligence,* 1**,** 3–55.

Mowshowitz, A. (1997) On the theory of virtual organization. *Systems research and behavioral science,* 14.

Müller, J. E., Pischel, M. & Thiel, M. (1995) Modelling reactive behaviour in vertically layered agent architectures. *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents.* Amsterdam, The Netherlands, Springer.

Mui, L., Halberstadt, A. & Mohtashemi, M. (2002) Notions of reputation in multi-agents systems: a review. *Proceedings of the First International Conference on Autonomous Agents and MAS.* Bologna, Italy.

Muñoz, C., Acevedo, P., Salvo, S., Fagalde, G. & Vargas, F. (2007) Forest fire detection using NOAA/16-LAC satellite images in the Araucanía Region, Chile. *Bosque,* 28**,** 119-128.

Mustafaraj, E. (2007) Knowledge Extraction and Summarization for Textual Case-Based Reasoning. *Fachbereich Mathematik und Informatik.* Philipps-Universitat Marburg.

Nagendra Prasad, M. V. & Lesser, V. R. (1999) Learning situation-specific coordination in cooperative multi-agent systems. *Autonomous agents and multi-agent systems,* 2**,** 173-207.

Nair, R., Tambe, M. & Marsella, S. (2003a) Role allocation and reallocation in multiagent teams: Towards a practical analysis. *Proceedings of Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-03).* ACM New York, NY, USA.

Nair, R., Tambe, M. & Marsella, S. (2003b) The role of emotions in multiagent teamwork: A preliminary investigation. *Who needs emotions: the brain meets the robot.* Oxford University Press.

Nakada, H., Sato, M. & Sekiguchi, S. (1999) Design and implementations of Ninf: towards a global computing infrastructure. *Future Generation Computer Systems,* 15**,** 649-658.

Natis, Y. & Schulte, R. (2003) Introduction to service-oriented architecture. *Gartner Group*.

Negroponte, N. (1996) *Being digital,* Random House Inc. New York, NY, USA.

Nelson, R. K., Kile, B. M., Plata, D. L., Sylva, S. P., Xu, L., Reddy, C. M., Gaines, R. B., Frysinger, G. S. & Reichenbach, S. E. (2006) Tracking the Weathering of an Oil Spill with Comprehensive Two-Dimensional Gas Chromatography. *Environmental Forensics,* 7**,** 33-44.

Newell, A. (1994) Reflections on the knowledge level. *Artificial Intelligence in Perspective,* 59**,** 31-38.

Ng, S. T. (2001) EQUAL: a case-based contractor prequalifier. *Automation in construction,* 10**,** 443-457.

Norman, T. J., Preece, A., Chalmers, S., Jennings, N. R., Luck, M., Dang, V. D., Nguyen, T. D., Deora, V., Shao, J. & Gray, W. A. (2004) Conoise: Agent-based formation of virtual organisations. *Knowledge-Based Systems,* 17**,** 103–111.

Nugent, C. & Cunningham, P. (2005) A Case-Based Explanation System for Black-Box Systems. *Artificial Intelligence Review,* 24**,** 163-178.

Nwana, H. S. & Ndumu, D. T. (1998) A brief introduction to software agent technology. *Agent Technology: Foundations, Applications, and Markets***,** 29–47.

Nwana, H. S., Ndumu, D. T., Lee, L. C. & Collis, J. C. (1999) ZEUS: a toolkit and approach for building distributed multi-agent systems. *In Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS '99).* Seattle, Washington, USA, ACM, New York, NY.

Omg (1995a) Compound Presentation and Compound Interchange Facilities, Part I, OMG Document 95-3-31. *Apple Computer, Inc. Component Integration Laboratories, Inc., International Business Machines Corporation, Novell Incorporated,.*

Omg (1995b) CORBA Services: Common Object Services Specification, Revised Edition. *Object Management Group.*

Omg (1996) Description of New OMA Reference Model, Draft 1. *Object Management Group.*

Omicini, A. (2001) SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems. *Agent-Oriented Software Engineering,* 1957**,** 185-193.

Ontañón, S. & Plaza, E. (2003) Collaborative Case Retention Strategies for CBR Agents. *Lecture Notes in Artificial Intelligence,* 21678**,** 394-405.

Ossowski, S. (1999) *Coordination in artificial agent societies: social structures and its implications for autonomous problem-solving agents*, Springer Verlag.

Ossowski, S. & García-Serrano, A. (1998) Social Co-ordination among Autonomous Problem-Solving Agents. IN WOBCKE, W., PAGNUCCO, M. & ZHANG, C. (Eds.) *In Proceedings of the Workshops on Commonsense Reasoning, intelligent Agents, and Distributed Artificial intelligence.* Lecture Notes In Computer Science, Springer-Verlag, London.

Palenzuela, J. M. T., Vilas, L. G. & Cuadrado, M. S. (2006) Use of ASAR images to study the evolution of the Prestige oil spill off the Galician coast. *International Journal of Remote Sensing,* 27**,** 1931-1950.

Panzarasa, P., Jennings, N. R. & Norman, T. J. (2002) Formalizing collaborative decision-making and practical reasoning in multi-agent systems. *Journal of logic and computation,* 12**,** 55-117.

Parker, L. E. (1993) Designing control laws for cooperative agent teams. *Proceedings of the IEEE International Conference on Robotics and Automation.*

Parunak, H. V. D. & Odell, J. (2002) Representing social structures in UML. *Agent-Oriented Software Engineering II, LNCS,* 2222.

Parunak, H. V. D., Savit, R. & Riolo, R. L. (1998) Agent-Based Modeling vs Equation-Based Modeling: A Case Study and Users' Guide. *Lecture Notes in Computer Science,* 1534**,** 10-25.

Pasley, J. (2005) How BPEL and SOA are changing Web services development. *IEEE Internet Computing,* 9**,** 60-67.

Patterson, D., Rooney, N., Dobrynin, V. & Galushka, M. (2005) Sophia: A novel approach for Textual Case-based Reasoning. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence.*

Pattison, H. E., Corkill, D. D. & Lesser, V. R. (1987) Instantiating descriptions of organizational structures. *Distributed artificial intelligence, Research Notes in Artificial Intelligence,* I**,** 59-96.

Pavón, R., Díaz, F., Laza, R. & Luzón, V. (2008) Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study. *Expert Systems With Applications,* 36**,** 3407-3420.

Peiró, J. M. (1995) Psicología de la Organización (I y II). *Madrid: UNED.*

Pérez, E. I., Coello, C. A. C. & Aguirre, A. H. (2005) Extraction and reuse of design patterns from genetic algorithms using case-based reasoning. *Soft Computing-A Fusion of Foundations, Methodologies and Applications,* 9**,** 44-53.

Periáñez, R. (2007) Chemical and oil spill rapid response modelling in the Strait of Gibraltar–Alborán Sea. *Ecological Modelling,* 207**,** 210-222.

Periáñez, R. & Pascual-Granged, A. (2008) Modelling surface radioactive, chemical and oil spills in the Strait of Gibraltar. *Computers and Geosciences,* 34**,** 163-180.

Plaza, E., Armengol, E. & Ontañón, S. (2005) The Explanatory Power of Symbolic Similarity in Case-Based Reasoning. *Artificial Intelligence Review,* 24**,** 145-161.

Pokahr, A., Braubach, L. & Lamersdorf, W. (2003) Jadex: Implementing a BDI-Infrastructure for JADE Agents. *In EXP - in search of innovation (Special Issue on JADE)***,** 76-85.

Polani, D. (2001) Measures for the organization of self-organizing maps. *Springer Studies In Fuzziness And Soft Computing Series***,** 13-44.

Policastro, C. A., Carvalho, A. C. & Delbem, A. C. B. (2006) Automatic knowledge learning and case adaptation with a hybrid committee approach. *Journal of Applied Logic,* 4**,** 26-38.

Polzlbauer, G. (2004) Survey and Comparison of Quality Measures for Self-Organizing Maps. *Proceedings of the Fifth Workshop on Data Analysis (WDA'04)***,** 67-82.

Popper, K. R. (1982) *The open universe: An argument for indeterminism*, Hutchinson, London.

Powell, W. W. (1991) Neither market nor hierarchy: Network forms of organization. *Markets, Hierarchies and Networks. The Coordination of Social Life***,** 265–276.

Prada, M. L. (2004) *Vivir al sol,* Oviedo, KRK Ediciones.

Price, J. M., Ji, Z. G., Reed, M., Marshall, C. F., Howard, M. K., Guinasso Jr, N. L., Johnson, W. R. & Rainey, G. B. (2003) Evaluation of an oil spill trajectory model using satellite-tracked, oil-spill-simulating drifters. *OCEANS 2003. Proceedings,* 3.

Pynadath, D. V. & Tambe, M. (2002) The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research,* 16**,** 389-423.

Qingling, Z. & Ying, L. (2007) Monitoring Marine Oil-spill Using Microwave Remote Sensing Technology. *Electronic Measurement and Instruments, 2007. ICEMI'07. 8th International Conference on***,** 4-69.

Ramchurn, S. D., Huynh, D. & Jennings, N. R. (2005) Trust in multi-agent systems. *The Knowledge Engineering Review,* 19**,** 1-25.

Rao, A. & Georgeff, M. P. (1992) An abstract architecture for rational agents. *Computational Intelligence,* 14**,** 392-429.

Reed, M., Ekrol, N., Rye, H. & Turner, L. (1999) Oil Spill Contingency and Response (OSCAR) Analysis in Support of Environmental Impact Assessment Offshore Namibia. *Spill Science and Technology Bulletin,* 5**,** 29-38.

Reed, S. & Lesser, V. R. (1980) Division of labor in honey bees and distributed focus of attention. *University of Massachusetts/Amherst Computer and Information Science Department Technical Report 80,* 17.

Reverte, J. (2006) *El médico de Ifni,* Barcelona, Areté.

Rivera De La Cruz, M. (2009) *La importancia de las cosas,* Barcelona, Planeta.

Rodríguez, J. A., Noriega, P., Sierra, C. & Padget, J. (1997) FM96. 5: A Java-based electronic auction house. *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97),*. London, UK.

Rodríguez, R., Cortés, A., Margalef, T. & Luque, E. (2008) An Adaptive System for Forest Fire Behavior Prediction. *11th IEEE International Conference on Computational Science and Engineering.*

Romelaer, P. (2002) Organization: a diagnosis method. *Cahier.* University Paris IX Dauphine, Crepa Laboratory.

Ros, F., Pintore, M. & Chrétien, J. R. (2007) Automatic design of growing radial basis function neural networks based on neighboorhood concepts. *Chemometrics and Intelligent Laboratory Systems,* 87**,** 231-240.

Ros, R., Veloso, M., De Mantaras, R. L., Sierra, C. & Arcos, J. L. (2006) Retrieving and Reusing Game Plays for Robot Soccer. *Advances in Case-Based Reasoning,* 4106**,** 2006.

Rosenberry, W., Kenney, D. & Fisher, G. (1992) *Understanding DCE,* O'Reilly & Associates, Inc. Sebastopol, CA, USA.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. & Edwards, D. D. (1995) *Artificial intelligence: a modern approach,* Prentice hall Englewood Cliffs, NJ.

Sabater, J. & Sierra, C. (2001) Social regret, a reputation model based on social relations. *ACM SIGecom Exchanges,* 3**,** 44-56.

Sahin, Y. G. (2007) Animals as Mobile Biological Sensors for Forest Fire Detection. *Sensors,* 7**,** 3084-3099.

Sánchez Dragó, F. (2009) *Soseki : inmortal y tigre,* Barcelona, Planeta.

Sandhlom, T. W. & Lesser, V. R. T. (1997) Coalitions among computationally bounded agents. *Artificial Intelligence,* 94**,** 99-137.

Sandholm, T. (2002) Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence,* 135**,** 1-54.

Sandholm, T. (2006) Optimal winner determination algorithms. *Combinatorial Auctions***,** 337–368.

Sandholm, T., Larson, K., Andersson, M., Shehory, O. & Tohmé, F. (1999) Coalition structure generation with worst case guarantees. *Artificial Intelligence,* 111**,** 209-238.

Sanz, G. (2002) Modelado de Sistemas Multi-Agente. *Departamento de Sistemas Informaticos y Programacion.* Universidad Complutense de Madrid.

Savater, F. (2008) *La hermandad de la buena suerte,* Barcelona, Planeta.

Scerri, P., Pynadath, D. & Tambe, M. (2002) Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research,* 17**,** 171-228.

Schmitt, J. B. & Roedig, U. (2005) Sensor Network Calculus--A Framework for Worst Case Analysis. *Lecture Notes in Computer Science,* 3560**,** 141-154.

Schroth, C. & Christ, O. (2007) Brave new web: Emerging design principles and technologies as enablers of a global soa. *Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007).* Salt Lake City, USA.

Schwaninger, M., Körner, M. & Institut Für, B. (2000) *A theory for optimal organization*, Institute of Management, University of St. Gallen (HSG).

Searle, J. R. (1969) *Speech acts: An essay in the philosophy of language*, Cambridge University Press.

Sen, S. (1996) Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. *Proc. of the Second International Conference on Multiagent Systems.* AAAI Press.

Serón, F. J., Gutiérrez, D., Magallón, J., Ferragut, L. & Asensio, M. I. (2005) The Evolution of a Wildland Forest Fire Front. *The Visual Computer,* 21**,** 152-169.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C. & Ram, A. (2007) Transfer learning in real-time strategy games using hybrid cbr/rl. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*.

Shehory, O. & Kraus, S. (1998) Methods for task allocation via agent coalition formation. *Artificial Intelligence,* 101**,** 165-200.

Shehory, O., Sycara, K., Chalasani, P. & Jha, S. (1998) Agent cloning: an approach to agent mobility and resource allocation. *IEEE Communications Magazine,* 36**,** 58-67.

Shen, J., Zhang, X. & Lesser, V. (2004) Degree of local cooperation and its implication on global utility. *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AA-MAS 2004)*. New York, IEEE Computer Society Washington, DC, USA.

Shen, W. & Norrie, D. H. (1998) A hybrid agent-oriented infrastructure for modeling manufacturing enterprises. *Knowledge Acquisition Workshop (KAW'98).*

Shiu, S. C. K. & Pal, S. K. (2004) Case-Based Reasoning: Concepts, Features and Soft Computing. *Applied Intelligence,* 21**,** 233-238.

Shoham, Y., Computer Science, D. & Stanford, U. (1997) Agent-oriented programming. *Knowledge Representation and Reasoning Under Uncertainty*. Springer.

Shoham, Y. & Tennenholtz, M. (1995) On social laws for artificial agent societies: Off-line design. *Artificial Intelligence,* 73**,** 231-252.

Sichman, J. S. & Demazeau, Y. (2001) On social reasoning in multi-agent systems. *Revista Iberoamericana de Inteligencia Artificial,* 13**,** 68-84.

Siegel, J. (1998) OMG overview: CORBA and the OMA in enterprise computing. *Communications of the ACM,* 41**,** 37-43.

Sierra, C., Sabater, J., Augusti, J. & Garcia, P. (2004) SADDE: Social agents design driven by equations. *Methodologies and software engineering for agent systems. Kluwer Academic Publishers*.

Simon, H. A. (1969) *The sciences of the artificial,* MIT press.

Sims, M., Corkill, D. & Lesser, V. (2004) Separating domain and coordination in multi-agent organizational design and instantiation. *Proceedings of the International Conference on Intelligent Agent Technology (IAT 20004).* Beijing, China.

Sims, M., Goldman, C. V. & Lesser, V. (2003) Self-organization through bottom-up coalition formation. *Proceedings of Second International Joint Conference on Au- tonomous Agents and MultiAgent Systems (AAMAS 2003).* ACM New York, NY, USA.

Smith, R. G. (1980) The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers,* 29**,** 1104-1113.

Snidaro, L. & Foresti, G. L. (2007) Knowledge representation for ambient security. *Expert Systems,* 24**,** 321-333.

So, Y. & Durfee, E. H. (1996) Designing tree-structured organizations for computational agents. *Computational & Mathematical Organization Theory,* 2**,** 219-245.

Soh, L. K. (2003) A satisficing, negotiated, and learning coalition formation architecture. *Distributed Sensor Networks: A multiagent perspective***,** 109–138.

Solberg, A. H. S., Brekke, C. & Husoy, P. O. (2007) Oil Spill Detection in Radarsat and Envisat SAR Images. *Geoscience and Remote Sensing, IEEE Transactions on,* 45**,** 746-755.

Solberg, A. H. S., Storvik, G., Solberg, R. & Volden, E. (1999) Automatic detection of oil spills in ERS SAR images. *IEEE Transactions on Geoscience and Remote Sensing,* 37**,** 1916-1924.

Song, X., Petrovic, S. & Sundar, S. (2007) A Case-Based Reasoning Approach to Dose Planning in Radiotherapy. *International Conference on Case-Based Reasoning, ICCBR 2007, Proceedings.*

Sørmo, F., Cassens, J. & Aamodt, A. (2005) Explanation in Case-Based Reasoning–Perspectives and Goals. *Artificial Intelligence Review,* 24**,** 109-143.

Spasic, I., Ananiadou, S. & Tsujii, J. (2005) MaSTerClass: a case-based reasoning system for the classification of biomedical terms. *Bioinformatics,* 21**,** 2748-2758.

Spread (2004) http ://www.adai.pt. *Forest Fire Spread Prevention and Mitigation.*

Stal, M., Technol, S. C. & Munich, G. (2006) Using architectural patterns and blueprints for service-oriented architecture. *IEEE software,* 23**,** 54-61.

Stammer, D., Wunsch, C., Giering, R., Eckert, C., Heimbach, P., Marotzke, J., Adcroft, A., Hill, C. N. & Marshall, J. (2003) Volume, heat, and freshwater transports of the global ocean circulation 1993–2000, estimated from a general circulation model constrained by World Ocean Circulation Experiment (WOCE) data. *Journal of Geophysical Research,* 108.

Stefanoiu, D., Ulieru, M. & Norrie, D. (2000) Fuzzy Modeling of Multi-Agent Systems Behavior. Vagueness Minimization. *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000).*

Stonebraker, M., Aoki, P. M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C. & Yu, A. (1996) Mariposa: A wide-area distributed

database system. *The VLDB Journal The International Journal on Very Large Data Bases,* 5**,** 48-63.

Subramanian, R. & Goodman, B. D. (2005) *Peer-to-peer computing: the evolution of a disruptive technology,* Idea Group Pub.

Sun, Z., Finnie, G. & Weber, K. (2004) Case base building with similarity relations. *Information Sciences,* 165**,** 21-43.

Sycara, K., Decker, K. & Williamson, M. (1997) Middle-agents for the internet. *Proceedings of the 15th International Joint Conference on Artificial Intelligence.*

Sycara, K., Paolucci, M., Van Velsen, M. & Giampapa, J. (2003) The retsina mas infrastructure. *Autonomous agents and multi-agent systems,* 7**,** 29-48.

Sycara, K. P. (1998a) The many faces of agents. *AI magazine,* 19**,** 11-12.

Sycara, K. P. (1998b) Multiagent systems. *AI magazine,* 19**,** 79-92.

Tambe, M. (1997) Towards flexible teamwork. *Journal of Artificial Intelligence Research,* 7**,** 83-124.

Tambe, M., Adibi, J., Al-Onaizan, Y., Erdem, A., Kaminka, G. A., Marsella, S. C. & Muslea, I. (1999) Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence,* 110**,** 215-239.

Tello, M., López-Martínez, C. & Mallorqui, J. J. (2006) A Novel Algorithm for Automatic Ship and Oil Spill Detection Based on Time-Frequency Methods. *Advances in SAR Oceanography from Envisat and ERS Missions, Proceedings of SEASAR* Frascati, Italy, European Space Agency.

Thatte, S. (2001) XLANG: Web services for business process design.

Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K. & Essiari, A. (1999) Certificate-based access control for widely distributed resources. *Proc. 8th Usenix Security Symposium.*

Tidhar, G., Heinze, C. & Selvestrel, M. (1998) Flying together: Modelling air mission teams. *Applied Intelligence,* 8**,** 195-218.

Tidhar, G., Rao, A. S. & Sonenberg, E. A. (1996) Guided team selection. *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96).* Kyoto, Japan.

Topouzelis, K., Karathanassi, V., Pavlakis, P. & Rokos, D. (2007) Detection and discrimination between oil spills and look-alike phenomena through neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing,* 62**,** 264-270.

Tsai, C. Y. & Chiu, C. C. (2007) A case-based reasoning system for PCB principal process parameter identification. *Expert Systems With Applications,* 32**,** 1183-1193.

Tse, L. (2007) *Tao Te Ching,* Integral.

Tsvetovat, M., Sycara, K., Chen, Y. & Ying, J. (2001) Customer coalitions in electronic markets. *Proceedings of the Fourth International Conference on Autonomous Agents.* Barcelona, Spain.

Tsvetovatyy, M., Gini, M., Mobasher, B. & Ski, Z. W. (1997) MAGMA: an agent based virtual market for electronic commerce. *Applied Artificial Intelligence,* 11**,** 501-523.

Tuler, S., Kay, R., Seager, T. P. & Linkov, I. (2006) Objectives and performance metrics in oil spill response: The Bouchard-120 and Chalk Point spill responses. *SERI Report.*

Turing, A. M. (1950) Computer machinery and intelligence. *Mind,* 59**,** 433-460.

Turner, R. M. (1993) The tragedy of the commons and distributed AI systems. *Dept. of Computer Science.* University of New Hampshire.

Ulieru, M. (2002) Emergence of Holonic Enterprises from Multi-Agent Systems: A Fuzzy-Evolutionary Approach. *Invited Chapter in Soft Computing Agents: A New Perspective on Dynamic Information Systems***,** 187-215.

Ulieru, M., Walker, S. & Brennan, B. (2001) Holonic enterprise as a collaborative information ecosystem. *In Proc. Workshop on Holons: Autonomous and Cooperating Agents for Industry.*

Valdivia, I. G. (1983) *Reflexiones en torno al orden social,* Editorial Jus.

Van Den Broek, E. L., Jonker, C. M., Sharpanskykh, A., Treur, J. & Yolum, P. (2006) Formal modeling and analysis of organizations. *Lecture Notes in Computer Science,* 3913**,** 18.

Van Steen, M. & Tanenbaum, A. S. (2002) *Distributed Systems: Principles and Paradigms,* Prentice Hall.

Vázquez-Salceda, J. & Dignum, F. (2003) Modelling Electronic Organizations. *Lecture Notes in Artificial Intelligence,* 2691**,** 584-593.

Vázquez-Salceda, J., Dignum, V. & Dignum, F. (2005) Organizing multiagent systems. *Autonomous agents and multi-agent systems,* 11**,** 307-360.

Verharen, E. (1997) A Language-Action Perspective on the Design of Cooperative Information Systems. *PhD Thesis.* Katholieke Universieit Brabant.

Vethamony, P., Sudheesh, K., Babu, M. T., Jayakumar, S., Manimurali, R., Saran, A. K., Sharma, L. H., Rajan, B. & Srivastava, M. (2007) Trajectory of an oil spill off Goa, eastern Arabian Sea: Field observations and simulations. *Environmental Pollution,* 148.

Vickrey, W. (1961) Counterspeculation, auctions, and competitive sealed tenders. *Journal of finance***,** 8-37.

Vinoski, S. (1993) Distributed object computing with CORBA. *C++ Report,* 5**,** 32-38.

Wagner, J. A. (2004) *Comportamiento organizativo: Consiguiendo la ventaja competitiva,* Thomson Learning Ibero.

Wagner, T. & Lesser, V. (2000) Relating quantified motivations for organizationally situated agents. *Lecture Notes in Computer Science,* 1757**,** 334-348.

Walker, A. & Wooldridge, M. (1995) Understanding the emergence of conventions in multi-agent systems. *Proceedings of the First International Conference on Multi–Agent Systems.* San Francisco, CA.

Watson, I. (1999) Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems,* 12**,** 303-308.

Watson, I. & Marir, F. (1994) Case-Based Reasoning: A Review. *The Knowledge Engineering Review,* 9**,** 327-354.

Wayner, P. (1995a) *Agents Unleashed: A public domain look at agent technology,* Academic Press Professional, Inc. San Diego, CA, USA.

Wayner, P. (1995b) Free Agents. *Byte,* March**,** 105-114.

Weber, M. (1978) Economy and Society (Berkeley. *University of California Press,* 1016**,** 308–338.

Weiser, M. (1993) Ubiquitous computing. *IEEE Computer,* 26**,** 71-72.

Weiss, G. (1999) Prologue: multiagent systems and distributed artificial intelligence. *Multiagent systems: a modern approach to distributed artificial intelligence, MIT Press, Cambridge, Massachusetts***,** 1-23.

Wellman, M. P. (1993) A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research,* 1**,** 1-23.

Wellman, M. P. (2004) Online marketplaces. *Practical Handbook of Internet Computing.* . Chapman Hall & CRC Press.

Wellman, M. P., Walsh, W. E., Wurman, P. R. & Mackie-Mason, J. K. (2001) Auction protocols for decentralized scheduling. *Games and Economic Behavior,* 35**,** 271-303.

Wellman, M. P. & Wurman, P. R. (1998) Market-aware agents for a multiagent world. *Robotics and autonomous Systems,* 24**,** 115-126.

Wellman, M. P. & Wurman, P. R. (1999) A trading agent competition for the research community. *Proceedings of the IJCAI-99 Workshop on Agent-Mediated Electronic Trading.*

Wiederhold, G. (1992) Mediators in the architecture of future information systems. *Computer Magazine,* 25**,** 38-49.

Wilson, D. C. (2001) Case-base Maintenance: The Husbandry of Experience. *PhD Thesis.* Indiana University.

Williamson, O. E. (1975) *Markets and hierarchies, analysis and antitrust implications: a study in the economics of internal organization,* Free Press.

Willmott, S., Dale, J., Burg, B., Charlton, P. & O'brien, P. (2001) Agentcities: a worldwide open agent network. *Agentlink News,* 8**,** 13-15.

Winograd, T. (1987) *Understanding computers and cognition: A new foundation for design,* Addison-Wesley.

Wirtz, K. W., Baumberger, N., Adam, S. & Liu, X. (2007) Oil spill impact minimization under uncertainty: Evaluating contingency simulations of the Prestige accident. *Ecological Economics,* 61**,** 417-428.

Wooldridge, M. (1999) Intelligent Agents. *in* WEISS, G. (Ed.) *Multiagent System: A Modern approach to Distributed artificial intelligence.* MIT press.

Wooldridge, M. (2002) *An Introduction to MultiAgent Systems,* Chichester, England, John Wiley & Sons.

Wooldridge, M. & Jennings, N. R. (1995) Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review,* 10**,** 115-152.

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000) The Gaia methodology for agent-oriented analysis and design. *Autonomous agents and multi-agent systems,* 3**,** 285-312.

Wooldridge, M. J. (2000) *Reasoning about rational agents,* MIT press.

Wooldridge, M. J. & Jennings, N. R. (1999) Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing,* 3**,** 20-27.

Wu, J. & Yu, Y. (2005) Connectionism-Based CBR Method for Distribution Short-Term Nodal Load Forecasting. *TENCON 2005. IEEE Region 10***,** 1-6.

Wurman, P. R., Wellman, M. P. & Walsh, W. E. (2001) A parametrization of the auction design space. *Games and Economic Behavior,* 35**,** 304-338.

Wybo, J. L., De Paris, E. M. & Antipolis, S. (1998) FMIS: a decision support system for forest fire prevention and fighting. *Engineering Management, IEEE Transactions on,* 45**,** 127-131.

Yadgar, O., Kraus, S. & Ortiz, C. (2003) Scaling up distributed sensor networks: cooperative large-scale mobile-agent organizations. *in* LESSER, V., ORTIZ, C. L. & TAMBE, M. (Eds.) *Distributed Sensor Networks: A Multiagent Perspective.*

Yang, B. S., Han, T. & Kim, Y. S. (2004) Integration of ART-Kohonen neural network and case-based reasoning for intelligent fault diagnosis. *Expert Systems With Applications,* 26**,** 387-395.

Yau, N. J. & Yang, J. B. (1998a) Applying case-based reasoning technique to retaining wall selection. *Automation in construction,* 7**,** 271-283.

Yau, N. J. & Yang, J. B. (1998b) Case-based reasoning in construction management. *Computer-Aided Civil and Infrastructure Engineering,* 13**,** 143-150.

Yu, W. & Liu, Y. (2006) Hybridization of CBR and numeric soft computing techniques for mining of scarce construction databases. *Automation in Construction,* 15**,** 33-46.

Zambonelli, F. (2002) Abstractions and infrastructures for the design and development of mobile agent organizations. *Lecture Notes in Computer Science,* 2222**,** 245-262.

Zambonelli, F., Jennings, N. R. & Wooldridge, M. (2001) Organizational abstractions for the analysis and design of multi-agent systems. *Lecture Notes in Computer Science***,** 235-252.

Zambonelli, F., Jennings, N. R. & Wooldridge, M. (2003) Developing multiagent systems: The Gaia methodology. *ACM Transactions on software Engineering and Methodology,* 12**,** 317-370.

Zhang, F., Ha, M. H., Wang, X. Z. & Li, X. H. (2004) Case adaptation using estimators of neural network. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics,* 4.

Zhang, X., Lesser, V. & Wagner, T. (2002) Integrative negotiation in complex organizational agent systems. *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003).* ACM New York, NY, USA.

Zhang, X. & Norrie, D. H. (1999) Holonic control at the production and controller levels. *Proc. 2nd Int. Workshop on Intelligent Manufacturing Systems.*

”Only strength can cooperate. Weakness can only beg.” *Dwight D. Eisenhower*

Appendix A. CORBA

CORBA is a mechanism in software for normalizing the method-call semantics between application objects that reside either in the same address space (application) or remote address space (same host, or remote host on a network). Version 1.0 was released in October 1991. CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. In this appendix, some technical specifications are explained, further and deeper developing the initial explanations given in previous chapter about CORBA.

The *Common Object Requesting Broker Architecture* (CORBA) is a standard defined by the Object Management Group(OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. One of the first specifications to be adopted by the OMG was the CORBA specification. It details the interfaces and characteristics of the ORB component of the OMA. As of this writing, the last major update of the CORBA specification was in mid-1995 when the OMG released CORBA 2.0 [OMG, 1996]. The main features of CORBA 2.0 are: *ORB Core, OMG Interface Definition Language (OMG IDL)*

Interface Repository, Language Mappings, Stubs and Skeletons, Dynamic Invocation and Dispatch, Object Adapters and *Inter-ORB Protocols.*

Most of these are illustrated in *figure 34*, which also shows how the components of CORBA relate to one another. Each component is described in detail below.

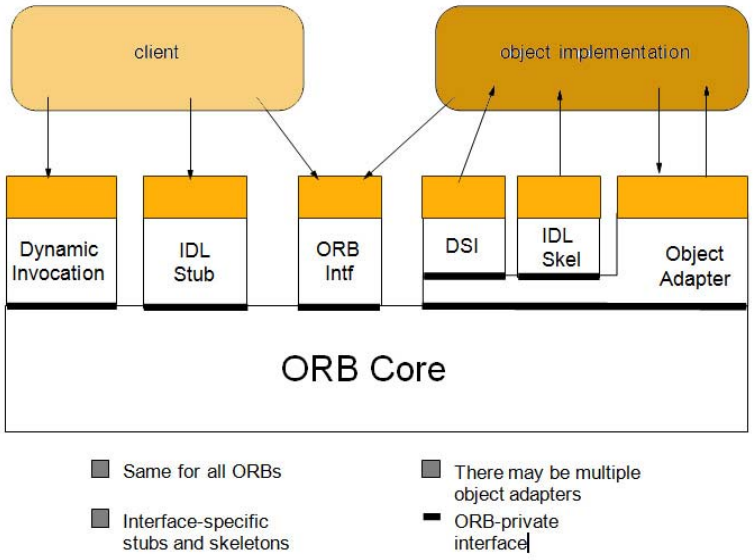


Figure 34. *Common Object Request Broker Architecture.*

A.1. ORB CORE

As mentioned above, the ORB delivers requests to objects and returns any responses to the clients making the requests. The object that a client wishes the ORB to direct a request to is called the target object. The key feature of the ORB is the transparency of how it facilitates client/object communication. Ordinarily, the ORB hides the following:

– *Object location:* The client does not know where the target object resides. It could reside in a different process on another machine across the network, on the same machine but in a different process,

or within the same process.

- *Object implementation:* The client does not know how the target object is implemented, what programming or scripting language it was written in, or the operating system (if any) and hardware it executes on.
- *Object execution state:* When it makes a request on a target object, the client does not need to know whether that object is currently activated (i.e., in an executing process) and ready to accept requests. The ORB transparently starts the object if necessary before delivering the request to it.
- *Object communication mechanisms:* The client does not know what communication mechanisms (e.g., TCP/IP, shared memory, local method call, etc.) the ORB uses to deliver the request to the object and return the response to the client.

These ORB features allow application developers to worry more about their own application domain issues and less about low-level distributed system programming issues.

To make a request, the client specifies the target object by using an object reference. When a CORBA object is created an object reference for it is also created. When used by a client, an object reference always refers to the same object for which it was created, for as long as that object still exists. In other words, an object reference only ever refers to one single object.

Object references are both immutable and opaque, so a client can't *"reach into"* the object reference and modify it. Only an ORB knows what's *"inside"* an object reference. Object references can have standardized formats, such as those for the OMG standard Internet Inter-ORB Protocol and Distributed Computing Environment Common Inter-ORB Protocol, or they can have proprietary formats.

A.2. OMG INTERFACE DEFINITION LANGUAGE (OMG IDL)

Before a client can make requests on an object, it must know the types of operations supported by the object. An object's interface specifies the operations and types that the object supports and thus defines the requests that can be made on the object. Interfaces for objects are defined in the *OMG Interface Definition Language* (*OMG IDL*). Interfaces are similar to classes in C++ and interfaces in Java.

An important feature of *OMG IDL* is its language independence. Since *OMG IDL* is a declarative language, not a programming language, it forces interfaces to be defined separately from object implementations. This allows objects to be constructed using different programming languages and yet still communicate with one another. Language-independent interfaces are important within heterogeneous systems, since not all programming languages are supported or available on all platforms.

OMG IDL provides a set of types that are similar to those found in a number of programming languages. It provides basic types such as *long*, *double*, and *boolean*, constructed types such as *struct* and discriminated *union*, and template types such as *sequence* and *string*. Types are used to specify the parameter types and return types for operations. As seen in the example above, operations are used within interfaces to specify the services provided by those objects that support that particular interface type. To define exceptional conditions that may arise during the course of an operation, *OMG IDL* provides exception definitions. Like structs, *OMG IDL* exceptions may have one or more data members of *any OMG IDL* type. The *OMG IDL* module construct allows for scoping of definition names to prevent name clashes.

A.3. LANGUAGE MAPPINGS

As mentioned before, *OMG IDL* is just a declarative language, not a full-fledged programming language. As such, it does not provide features like control constructs, nor is it directly used to implement distributed applications. Instead, language mappings determine how *OMG IDL* features are mapped to the facilities of a given programming language.

At the time of this writing, the OMG has standardized language mappings for C, C++, Smalltalk, and Ada 95. Likewise, mappings for the UNIX Bourne shell and for COBOL are nearing completion. A mapping for the Java language is just beginning, but is slated to finish quickly keeping up with the high demand for Java/CORBA integration. Language mappings for other languages such as Perl, Eiffel, and Modula-3 have also been written by various interested parties, but have not been submitted to the OMG for approval.

To understand what a language mapping contains, consider the mapping for the C++ language. Not surprisingly, OMG IDL interfaces map to C++ classes, with operations mapping to member functions of those classes. Object references map to objects that support the operator-> function (i.e., either a normal C++ pointer to an interface class, or an object instance with an overloaded operator->). Modules map to C++ namespaces (or to nested classes for C++ compilers that do not yet support namespaces).

Another important aspect of an OMG IDL language mapping is how it maps the ORB interface and other pseudo-objects that are found in the CORBA specification. Pseudo-objects are ORB interfaces that are not implicitly derived from CORBA::Object, such as the ORB itself. In other words, pseudo-objects are not real CORBA objects, but specifying such interfaces just like normal object interfaces are specified allows applications to manipulate the ORB much like they manipulate normal objects.

A third important part of any language mapping specification is how CORBA objects are implemented in the language. In object-oriented languages such as Java, Smalltalk, and C++, for example, CORBA objects are implemented as programming language objects. In C, objects are written as abstract data types. For instance, a typical implementation consists of a struct that holds the state of the object and a group of C functions (which correspond to the OMG IDL operations supported by the object) to manipulate that state.

OMG IDL language mappings are where the abstractions and concepts specified in CORBA meet the "real world" of implementation. Thus, their importance for CORBA applications cannot be overstated. A poor or incomplete mapping specification for a given language results in programmers being unable to effectively utilize CORBA technology in that language. Language mapping specifications are therefore always undergoing periodic improvement in order to incorporate evolution of programming languages, as well as to add features that fulfil new requirements discovered by writing new applications.

A.4. INTERFACE REPOSITORY

Every CORBA-based application requires access to the OMG IDL type system when it is executing. This is necessary because the application must know the types of values to be passed as request arguments. In addition, the application must know the types of interfaces supported by the objects being used.

Many applications require only static knowledge of the OMG IDL type system. Typically, an OMG IDL specification is compiled or translated into code for the application's programming language by following the translation rules for that language, as defined by its language mapping. Then, this generated code is built directly into the application.

With this approach, the application's knowledge of the OMG IDL type system is fixed when it is built.

If the type system of the rest of the distributed system ever changes in a way that is incompatible with the type system built into the application, the application must be rebuilt. For example, if a client application depends on the Factory interface, and the name of the create operation in the Factory interface is changed to create object, the client application will have to be rebuilt before it can make requests on any Factory objects.

There are some applications, however, for which static knowledge of the OMG IDL type system is impractical. For example, consider a Gateway that allows applications in a foreign object system (such as Microsoft Component Object Model (COM) applications) to access CORBA objects. Having to recompile and rebuild the Gateway every time someone added a new OMG IDL interface type to the system would result in a very difficult management and maintenance problem. Instead, it would be much better if the Gateway could dynamically discover and utilize type information as needed.

The CORBA Interface Repository (IR) allows the OMG IDL type system to be accessed and written programmatically at runtime. The IR is itself a CORBA object whose operations can be invoked just like any other CORBA object. Using the IR interface, applications can traverse an entire hierarchy of OMG IDL information.

For example, an application can start at the top-level scope of the IR and iterate over the entire module definitions defined there. When the desired module is found, it can open it and iterate in a similar manner over all the definitions inside it. This hierarchical traversal approach can be used to examine all the information stored within an IR.

A.5. STUBS AND SKELETONS

In addition to generating programming language types, OMG IDL language compilers and translators also generate client-side stubs and server-side skeletons. A stub is a mechanism that effectively creates and issues requests on behalf of a client, while a skeleton is a mechanism that delivers requests to the CORBA object implementation. Since they are translated directly from OMG IDL specifications, stubs and skeletons are normally interface-specific.

Dispatching through stubs and skeletons is often called static invocation. OMG IDL stubs and skeletons are built directly into the client application and the object implementation. Therefore, they both have complete a priori knowledge of the OMG IDL interfaces of the CORBA objects being invoked.

Language mappings usually map operation invocation to the equivalent of a function call in the programming language. Once the request arrives at the target object, the server ORB and the skeleton cooperate to unmarshal the request (convert it from its transmissible form to a programming language form) and dispatch it to the object. Once the object completes the request, any response is sent back the way it came: through the skeleton, the server ORB, over the connection, and then back through the client ORB and stub, before finally being returned to the client application.

This description shows that stubs and skeletons play important roles in connecting the programming language world to the underlying ORB. In this sense they are each a form of the Adapter and Proxy patterns [Gamma *et al.*, 1995]. The stub adapts the function call style of its language mapping to the request invocation mechanism of the ORB. The skeleton adapts the request dispatching mechanism of the ORB to the upcall method form expected by the object implementation.

A.6. DYNAMIC INVOCATION AND DISPATCH

In addition to static invocation via stubs and skeletons, CORBA supports two interfaces for dynamic invocation:

- *Dynamic Invocation Interface (DII)* – which supports dynamic client request invocation;
- *Dynamic Skeleton Interface (DSI)* – which provides dynamic dispatch to objects.

The *DII* and the *DSI* can be viewed as a generic stub and generic skeleton, respectively. Each is an interface provided directly by the ORB, and neither is dependent upon the *OMG IDL* interfaces of the objects being invoked.

A.6.1. DYNAMIC INVOCATION INTERFACE

Using the DII, a client application can invoke requests on any object without having compile-time knowledge of the object's interfaces. For example, consider the foreign object Gateway described above. When an invocation is received from the foreign object system, the Gateway must turn that invocation into a request dispatch to the desired CORBA object. Recompiling the Gateway program to include new static stubs every time a new CORBA object is created is impractical. Instead, the Gateway can simply use the DII to invoke requests on any CORBA object. The DII is also useful for interactive programs such as browsers that can obtain the values necessary to supply the arguments for the object's operations from the user.

Currently, CORBA applications that require the ability to invoke requests using something other than a synchronous or one-way model must use the DII. This is because the deferred synchronous request invocation

capability is currently only provided by the DII. However, this restriction will soon be removed. Recently, the OMG issued an RFP for an Asynchronous Messaging Service that should result in the adoption of technology for higher-level communications models, such as store-and-forward services for the ORB. This RFP also requests technology for supporting deferred synchronous request invocation via static stubs.

While the DII offers more flexibility than static stubs, users of the DII should also be sure they are aware of its hidden costs [Vinoski, 1993, Gokhale and Schmidt, 1996]. In particular, creating a DII request may cause the ORB to transparently access the IR to obtain information about the types of the arguments and return value. Since the IR is itself a CORBA object, each transparent IR request made by the ORB could in fact be a remote invocation. Thus, the creation and invocation of a single DII request could in fact require several actual remote invocations, making a DII request several times more costly than an equivalent static invocation. Static invocations do not suffer from the overhead of accessing the IR since they rely on type information already compiled into the application.

A.6.2. DYNAMIC SKELETON INTERFACE

Analogous to the DII is the server-side Dynamic Skeleton Interface (DSI). Just as the DII allows clients to invoke requests without having access to static stubs, the DSI allows servers to be written without having skeletons for the objects being invoked compiled statically into the program.

The foreign object Gateway described above is a good example of an application that requires DSI functionality. A bidirectional Gateway must be able to act as both a client and a server – it must translate requests from the foreign object system into requests on CORBA objects, and turn requests from CORBA applications into foreign object invocations. As mentioned above, it can use the DII when it wants to act as a client. To act as a server, however, it

needs a server-side equivalent of the DII, allowing it to accept requests without requiring static skeletons for each object's interface type to be compiled into it. Requiring the Gateway to be recompiled each time a new OMG IDL interface was introduced into the CORBA side of the system would not work well in practice.

Unlike most of the other CORBA subcomponents, which were part of the initial CORBA specification, the DSI was only introduced at CORBA 2.0. The main reason for its introduction was to support the implementation of gateways between ORBs utilizing different communications protocols. Even though inter-ORB protocols were also introduced at CORBA 2.0, it was thought by some at the time that gateways would become the method of choice for ORB interoperation. Given that most commercially-available ORB systems already support the standard Internet Inter-ORB Protocol (IIOP), this prediction does not appear to have come true. Still, the DSI is a useful feature for a certain class of applications, especially for bridges between ORBs and for applications that serve to bridge CORBA systems to non-CORBA services and implementations.

A.7. OBJECT ADAPTERS

The final subcomponent of CORBA, the Object Adapter (OA), serves as the glue between CORBA object implementations and the ORB itself. As described by the Adapter pattern [Gamma *et al.*, 1995], an object adapter is an object that adapts the interface of another object to the interface expected by a caller. In other words, it is an interposed object that uses delegation to allow a caller to invoke requests on an object even though the caller does not know that object's true interface. *Figure 35* illustrates the role of an object adapter.

Object adapters represent another aspect of the effort to keep the ORB as simple as possible. Responsibilities of object adapters include:

- *Object registration* – OAs supply operations that allow programming language entities to be registered as implementations for CORBA objects. Details of exactly what is registered and how the registration is accomplished depends on the programming language.
- *Object reference generation* – OAs generate object references for CORBA objects.
- *Server process activation* – If necessary, OAs start up server processes in which objects can be activated.
- *Object activation* – OAs activate objects if they are not already active when requests arrive for them.
- *Request demultiplexing* – OAs must cooperate with the ORB to ensure that requests can be received over multiple connections without blocking indefinitely on any single connection.
- *Object upcalls* – OAs dispatch requests to registered objects.

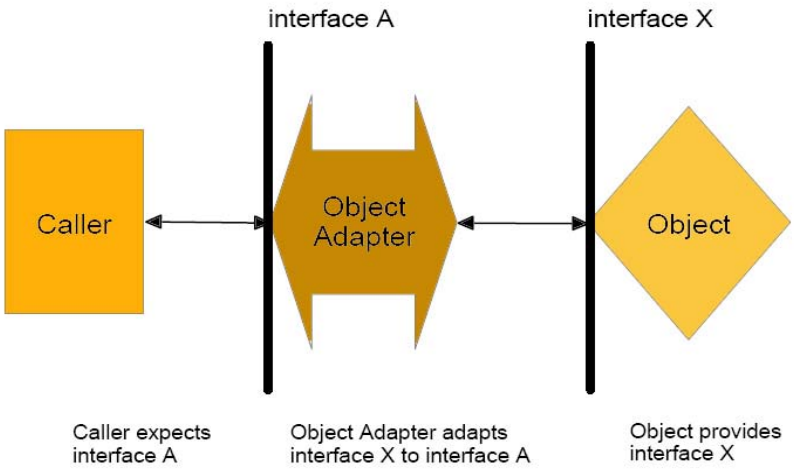


Figure 35. *Role of an Object Adapter.*

Without object adapters, the ability of CORBA to support diverse object implementation styles would be severely compromised. The lack of an object adapter would mean that object implementations would connect themselves directly to the ORB to receive requests. Having a standard set of just a few object upcall interfaces would mean that only a few styles of object implementation could ever be supported. Alternatively, standardizing many object upcall interfaces would add unnecessary size and complexity to the ORB itself.

A.8. INTER-ORB PROTOCOLS

Before CORBA 2.0, one of the biggest complaints about commercial ORB products is that they did not interoperate. Lack of interoperability was caused by the fact that the CORBA specification did not mandate any particular data formats or protocols for ORB communications. The main reason that CORBA did not specify ORB protocols prior to CORBA 2.0 was simply that interoperability was not a focus of the OMG at that time.

CORBA 2.0 introduced a general ORB interoperability architecture that provides for direct ORB-to-ORB interoperability and for bridge-based interoperability. Direct interoperability is possible when two ORBs reside in the same domain – in other words, they understand the same object references, the same OMG IDL type system, and perhaps shares the same security information. Bridge-based interoperability is necessary when ORBs from separate domains must communicate. The role of the bridge is to map ORB-specific information from one ORB domain to the other.

The general ORB interoperability architecture is based on the General Inter-ORB Protocol (GIOP), which specifies transfer syntax and a standard set of message formats for ORB interoperation over any connection-oriented transport. GIOP is designed to be simple and easy to implement while still allowing for reasonable scalability and performance.

The Internet Inter-ORB Protocol (IIOP) specifies how GIOP is built over TCP/IP transports. In a way, the relationship between IIOP and GIOP is somewhat like the relationship between an object's OMG IDL interface definition and its implementation. GIOP specifies protocol, just as an OMG IDL interface effectively defines the protocol between an object and its clients. IIOP, on the other hand, determines how GIOP can be implemented using TCP/IP, just as an object implementation determines how an object's interface protocol is realized. For a CORBA 2.0 ORB, support for GIOP and IIOP is mandatory.

The ORB interoperability architecture also provides for other environment-specific inter-ORB protocols (ESIOPs). ESIOPs allow ORBs to be built for special situations in which certain distributed computing infrastructures are already in use. The first ESIOP, which utilizes the Distributed Computing Environment (DCE) [Rosenberry *et al.*, 1992], is called the DCE Common Inter-ORB Protocol (DCE-CIOP). It can be used by ORBs in environments where DCE is already installed. This allows the ORB to leverage existing DCE functions, and it allows for easier integration of CORBA and DCE applications. Support for DCE-CIOP or any other ESIOP by a CORBA 2.0 ORB is optional.

" Science is organized knowledge. Wisdom is organized life." *Immanuel Kant*

Appendix B. Taxonomy of Organizations

Organizations represent the inner structure of the architecture proposed in this document. In this appendix, a complete classification of the organizations of agents is done. These include hierarchies, holarchies, coalitions, teams, congregations, societies, federations, markets, and matrix organizations. A description of each will be provided, discussing their advantages and disadvantages, and providing examples of how they may be instantiated and maintained.

In the fourth chapter of this document, organizations of agents have been explained, as an evolution of multi-agent systems. The organizations of agents represent a logic evolution of the multi-agent systems, introducing an internal organizational element that gives the organizations an upper point of view. The fact that the agents can work

together, with common objectives, sharing processes and interchanging information is quite useful when a system must connect different users or different services that may be located far from each other. Thus, the organizational capabilities of the agents allow the system that employ this methodology to structure the information and the objectives of the systems developed, improving the results and creating a kind of specialization in the tasks performed by the agents.

In this appendix, a complete taxonomy of the organizations of agents is done, explaining the different possibilities of organizations. For all the organizations exposed here, their main characteristics are explained as well as the formation techniques in order to create an organization of agents of a specific type.

B.1. HIERARCHIES

The hierarchy or hierarchical organization is perhaps the earliest example of structured, organizational design applied to multi-agent system and earlier distributed artificial intelligence architectures [Fox, 1979, Lesser and Erman, 1980, Davis and Smith, 1980, Bond and Gasser, 1988, Malone and Smith, 1988, Montgomery and Durfee, 1993]. Agents are conceptually arranged in a tree-like structure, as seen in *figure 36*, where agents higher in the tree have a more global view than those below them. In its strictest interpretation, interactions do not take place across the tree, but only between connected entities. More recent work [Mathieu *et al.*, 2002] has explored starting with a strict hierarchy and augmenting it with cross links to allow more direct communication, which can reduce some of the latency that results from repeated traversals up and down the tree.

The data produced by lower-level agents in a hierarchy typically travels upwards to provide a broader view, while control flows downward as the higher level agents provide direction to those below [Bond and Gasser,

1988]. The simplest instance of this structure consists of a two-level hierarchy, where the lower level agents' actions are completely specified by the upper, which produces a global view from the resulting information [Chandrasekaran, 1981]. More complex instances have multiple levels, while data flow, authority relations or other organizationally-dictated characteristics may not be absolute.

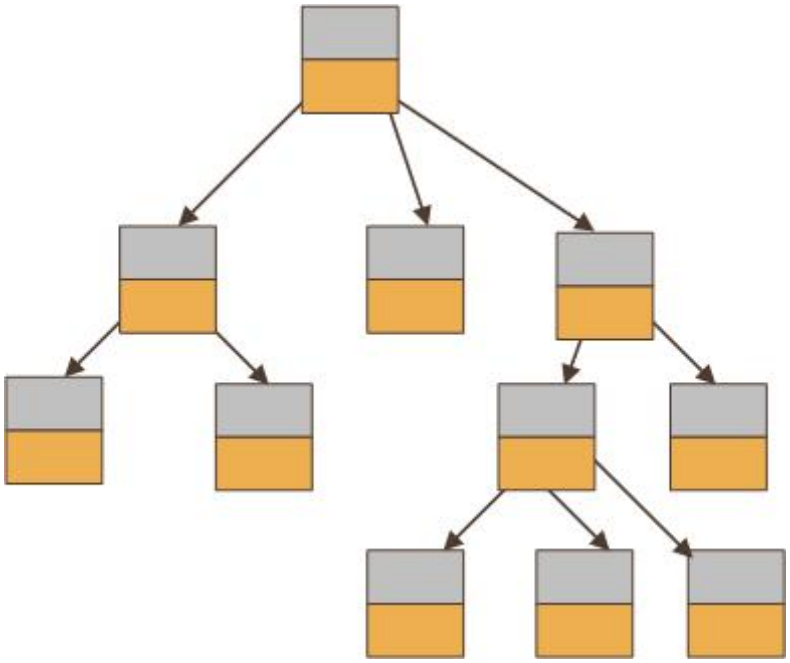


Figure 36. *Hierarchical organization.*

Fox [Fox, 1979] describes several different types of organizational hierarchies. The simple hierarchy endows a single apex member with the decision making authority in the system. Uniform hierarchies distribute this authority in different areas of the system to achieve efficiency gains through locality. Decisions are made by the agents which have both the information needed to reason about the decision, and the organizational authority to do make the decision. Each level acts as a filter, explicitly transferring information and implicitly transferring decisions up the hierarchy only when

necessary. Multi-divisional hierarchies further exploit localization by dividing the organization along "product" lines, where products might represent different physical artefacts, services, or high-level goals. Each division has complete control over their product, which facilitates the decision making and resource allocation process by limiting outside influences. The divisions themselves may still be organized under a higher-level entity which evaluates their performance and offers guidance, but is strictly separated from the divisional decision process.

B.1.1. CHARACTERISTICS

The applicability of hierarchical structuring comes from the natural decomposition possible in many different task environments. Indeed, task decomposition trees are a popular way of modelling individual agent plan recipes [Decker, 1996]; a hierarchical organization can be thought of as an assignment of roles and interconnections inspired by the global goal tree. The hierarchy's efficiency is also derived from this notion of decomposition, because the divide-and-conquer approach it engenders allows the system to use larger groups of agents more efficiently and address larger scale problems [Yadgar *et al.,* 2003]. This type of organization can constrain agents to a number of interactions that is small relative to the total population size. This allows control actions and behaviour decisions become more tractable, increased parallelism can be exploited, and because there is less potentially distracting data they can obtain a more cohesive view of the information pertinent to those decisions [Montgomery and Durfee, 1993].

It is not sufficient to simply aggregate increasing amounts of information to obtain higher utility or better performance. This information must be matched with sufficient computational power and analysis techniques to make effective use of the information [Lesser, 1991]. Without this, the effort to transfer the data may be wasted and the excess information distracts

the agent from more important tasks. Alternatively, the information can be summarized, approximated or otherwise processed on its way up the tree to reduce the information load. However, in doing so, a new dimension of uncertainty is introduced because of the potential for necessary details to be lost. In this situation, the decision making authority should be correctly placed within the structure to maximize the tractable amount of useful information that is available that retains an acceptable level of uncertainty or imprecision [Fox, 1979, Lesser and Corkill, 1981].

Using a hierarchy can also lead to an overly rigid or fragile organization, prone to single-point failures with potentially global consequences [Maturana *et al.*, 1999]. For example, if the apex agent were to fail the entire structure's cohesion could be compromised. Of course this agent could be replaced, but it may then prove costly to restore the concentrated information possessed by its predecessor. It is similarly susceptible to bottleneck effects if the scope of control decisions or data receipt is not effectively managed – consider what would happen if that apex agent received all the raw data produced by a large group of agents below it.

B.1.2. FORMATION

Although the algorithm itself does not enforce a strict hierarchy such as the one described earlier, Smith's contract net protocol [Smith, 1980, Davis and Smith, 1980] provides a straightforward mechanism to construct a series of connections with most of the same characteristics. In some of this early contract net work, the protocol was to explicitly form long-term organizational relationships, rather than the short-term contracts it has been typically used for more recently.

The hierarchical structure that is produced by the process is implicitly based on the way the high-level goal can be decomposed. Upon receipt of a new task, an agent first chooses to perform the task itself, or search for agents

willing to help complete the task. As part of this search process, the agent may decompose the task into subtasks or *contracts*. The agent, acting as a contractor, announces these contracts along with a bid specification to a subset of its peers who then decide if they wish to submit a bid. The bids which return to the contractor contain relevant information about the potential employee which allows it to discriminate among competing offers. An employee is selected and notified. Upon receipt of the new task, the employee now faces the same question – should it perform the task itself or contract it out? Repeated invocations of this process produce a hierarchy of contractors and employees.

Because agents individually choose which contracts to bid on, and contractors choose which bids to accept, this strategy can effectively assign tasks among a population of agents without the need for a global view. The drawback to this approach is that it is myopic. Because the contracting agent does not necessarily take into account the needs of other contractors, it may bind scarce resource in suboptimal ways. For example, it may select a particular bid when viable alternatives exist, even though that particular bidder is critical to another agent [Sims *et al.*, 2003].

As with most organizational structures, the shape of the hierarchy can affect the characteristics of both global and local behaviours. A very flat hierarchy where agents have a high degree of connectivity can lead to overloading if agent resources are both limited and consumed as a result of these connections. Conversely, a very tall structure may slow the system's performance because of the delays incurred by passing information across multiple levels. One approach to making this trade-off is the use of agent *cloning* [Ishida *et al.*, 1992, Decker *et al.*, 1997, Maturana *et al.*, 1999].

An agent in such a system may opt to create a copy or clone of itself, possessing the same capabilities as the original, in response to overloaded conditions. If additional resources are available for this clone to use, this

process allows the agent to dynamically create an assistant that can relieve excess burden from the original, reducing load-related errors or inefficiencies in the process. If the new agent is subordinate to the original, then a hierarchical organization will be formed in the process. Shehory [Shehory *et al.*, 1998] discusses using cloning when other task-reallocation strategies are not viable.

In this work, an agent's overall load is a function of its local processing, free memory and communication. It uses a dynamic programming technique to compute an optimal time to clone, and an appropriately idle computational node to house the new agent. The clone receives a subset of the original task(s). The clones themselves require resources, and the results they produce may require an additional hop to get to their ultimate destination, so they may also be merged or destroyed when these costs outweigh their benefits.

B.2. HOLARCHIES

The term *holon* was first coined by Arthur Koestler in his book The Ghost In The Machine [Koestler, 1968]. In this work, Koestler attempts to present a unified, descriptive theory of physical systems based on the nested, self-similar organization that many such systems possess. For example, biological, astrological and social systems are all comprised of multi-levelled, grouped hierarchies. A universe is comprised of a number of galaxies, which are comprised of a number of solar systems, and so on, all the way down to subatomic particles.

Each grouping in these systems has a character derived but distinct from the entities that are members of the group. At the same time, this same group contributes to the properties of one or more groups above it. The structure of each of these groupings is a basic unit of organization that can be seen throughout the system as a whole. Koestler called such units holons, from

the Greek word holos, meaning *"whole"*, and on, meaning *"part"*. Each holon exists simultaneously as both a distinct entity built from a collection of subordinates and as part of a larger entity.

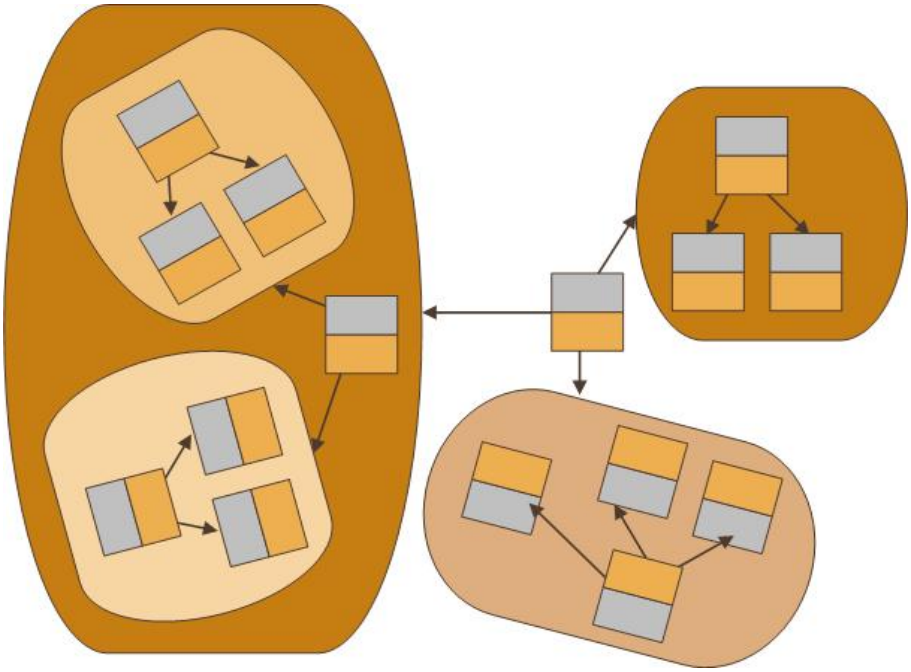


Figure 37. *Holarchical organization.*

True to Koestler's intent, this notion of a hierarchical, nested structure does accurately describe the organization of many systems. This concept has been exploited, primarily in business and manufacturing domains, to define and build structures called holarchies or holonic organizations which have this dual-nature characteristic. A sample such organization is shown in *figure 37*. In this diagram, hierarchical relationships are represented as directed edges, while circles represent holon boundaries.

Enterprises, companies, divisions, working groups and individuals can each be viewed as a holons taking part in a larger holarchy. Fischer [Fischer, 1999], Zhang [Zhang and Norrie, 1999], and Ulieru [Ulieru *et al.*, 2001] have each organized agent systems by modelling explicit or implied

divisions of labour in real-world systems as holons. In doing so, they create abstractions of these divisions, imparting capabilities to individual holons instead of individual agents. This layer of abstraction allows other entities in the system to make more effective use of these capabilities, by reasoning and interacting with the group as a single functional unit.

The defining characteristic of a holarchy is the partially-autonomous holon. Each holon is composed of one or more subordinate entities, and can be a member of one or more superordinate holons. Holons frequently have both a software and physical hardware component (Zhang and Norrie, 1999; Ulieru, 2002), although this does not preclude their usage in purely computational domains. The degree of autonomy associated with an individual holon is undefined, and could differ between levels or even between similar holons at the same level.

There is the presumption, however, that the level of autonomy is neither complete nor completely absent, as these extremes would lead to either a strict hierarchy or an unorganized grouping, respectively. Within the holarchy, the chain of command generally goes up – that is, subordinate holons relinquish some of their autonomy to the superordinate groupings they belong to.

However, there is also the more admitted notion that individual holons determine how to accomplish the tasks they are given, since they are likely the locus of relevant expertise. Many holonic structures also support connections between holons across the organization, which can result in more amorphous, web-like organizational structures that can change shape over time [Fischer, 1999, Zhang and Norrie, 1999].

It would not be incorrect to conclude that a holarchy is just a particular type of hierarchy. Relaxing the definition of hierarchy to allow some amount of cross-tree interactions and local autonomy, the two styles share many of the same features and can be used almost interchangeably. These richer models

then begin to resemble and take on the characteristics of nearly-decomposable hierarchies [Simon, 1969], where lateral interactions are weak but still relevant. Very flat holarchies can also begin to resemble federations, which will be discussed next.

B.2.1. CHARACTERISTICS

As with the conventional hierarchies explained before, holarchies are more easily applied to domains where goals can be recursively decomposed into subtasks that can be assigned to individual holons (although this is not essential). Given such decomposition, or a capability map of the population, the benefits the holonic organizations provide are derived primarily from the partially autonomous and encapsulated nature of holons. Holons are usually endowed with sufficient autonomy to determine how best to satisfy the requests they receive. Because the requester need not know exactly how the request will be completed, the holon potentially has a great deal of flexibility in its choice of behaviours, which can enable it to closely coordinate potentially complementary or conflicting tasks.

This characteristic reduces the knowledge burden placed on the requester and allows the holon's behaviour to adapt dynamically to new conditions without further coordination, so long as the original commitment's requirements are met. A drawback to this approach is that, lacking such knowledge, it is difficult to make predictions about the system's overall performance [Bongaerts, 1998].

B.2.2. FORMATION

The challenge in creating a holonic organization revolves around selecting the appropriate agents to reside in the individual holons. The purpose of the holon must be useful within the broader context of the organization's high-level goals, and the holon's members must be effective at satisfying that

purpose. Zhang [Zhang and Norrie, 1999] uses a model of static holons along with so-called mediator holons to create and adapt the organization. The static groups consist of product, product model and resource holons, each of which corresponds to a group of physical or information objects in the environment (e.g. manufacturing device, design plans, conveyors, etc.). The mediator holon ties these together, by managing orders, finding product data and coordinating resources in a manner similar to a federation, which will be discussed next. Each new task is represented by a dynamic mediator holon (DMH), which is created by the mediator holon. The DMH is destroyed when the task is completed.

Another approach to holarchy construction uses *fuzzy entropy* minimization to guide the formation of individual holonic clusters [Stefanoiu *et al.*, 2000, Ulieru, 2002]. In this work, the collection of holons is assumed to be initially described with a set of source-plans, each of which describes a potential assignment of holons to clusters, along with a set of probabilities that describe the degree of occurrence of those clusters. From this initial uncertain information, one can derive the preferences which agents have to work with one another, and then choose the source plan which has the minimal entropy with respect to those preferences.

The goal of this technique is to ensure that each holon has the necessary knowledge and expertise needed to perform its task. The preference that one agent has for another represents this knowledge or expertise requirement, so the minimally fuzzy set will satisfy this goal by clustering agents which have common preferences. In [Ulieru, 2002], Ulieru adds a genetic algorithm approach to this scheme to help explore the space of possible clustering assignments.

B.3. COALITIONS

The notion of a coalition of individuals has been studied by the game theory community for decades, and has proved to be a useful strategy in both real-world economic scenarios and multi-agent systems. Viewing the population of agents A as a set, then each subset of A is a potential coalition. Coalitions in general are goal-directed and short-lived; they are formed with a purpose in mind and dissolve when that need no longer exists, the coalition ceases to suit its designed purpose, or critical mass is lost as agents depart. Related research has extended this to longer-term agreements based on trust [Breban and Vassileva, 2001] and to the iterative formation of multiple coalitions in response to a dynamic task environment [Mérida-Campos and Willmott, 2004]. They may form in populations of both cooperative and self-interested agents.

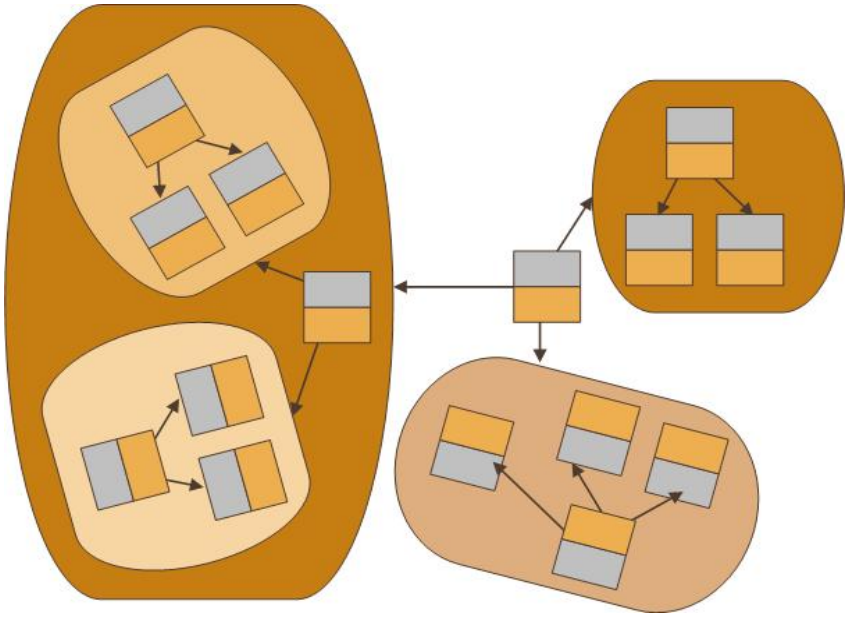


Figure 38. *Coalition-based organization.*

A population of agents organized into coalitions is shown in *figure 38*. Within a coalition, the organizational structure is typically flat, although there may be a distinguished "leading agent" which acts as a representative and intermediary for the group as a whole [Klusch and Gerber, 2002]. Once formed, coalitions may be treated as a single, atomic entity. Therefore, although coalitions have no explicit hierarchical characteristic, it is possible to form such an organization by nesting one group inside another.

Overlapping coalitions are also possible [Shehory and Kraus, 1998]. The agents in this group are expected to coordinate their activities in a manner appropriate to the coalition's purpose. Coordination does not take place among agents in separate coalitions, except to the degree that their individual goals interact. For example, if one coalition's goal depends on the results of another, these two groups might need to agree upon a deadline by which those results are produced. In this case, it would be the leading or representative agents forming the commitment, not arbitrary members of the coalition.

In addition to the problem of generating coalition structures, one must also determine how to solve the goal presented to the coalition. If the population is self-interested, a division of value to be apportioned to participants once that goal has been satisfied must also be generated and agreed upon [Sandhlom and Lesser, 1997].

B.3.1. CHARACTERISTICS

The motivation behind the coalition formation is the notion that the value of at least some of the participants may be super-additive along some dimension. Analogously, participants' costs may be sub-additive. This implies that utility can be gained by working in groups – this is the same rationale behind buying clubs, co-ops, unions, public protests and the *"safety in numbers"* principle. For instance, in an economic domain, a larger group of agents might have increased bargaining strength or other monetary reward

[Tsvetovat *et al.*, 2001].

In computational domains more efficient task allocation is expected, or the ability to solve goals with requirements greater than any single agent can offer [Shehory and Kraus, 1998]. In physically-limited systems, coalitions have been used to trade off the scope of agent interactions with the effectiveness of the system as a whole [Sims *et al.*, 2003]. This last application directly affects the coordination costs incurred by the system.

It can be argued that all agents in the environment should always join to form the all-inclusive *grand coalition*. Indeed, under certain circumstances this is appropriate, since the structure would have the resources of all available agents at its disposal, which theoretically would provide the maximum value. There are costs associated with forming and maintaining such a structure however, and in real world scenarios this can be both an impractical and unnecessarily coarse solution [Sandhlom and Lesser, 1997].

Therefore, the problem of coalition formation becomes one of selecting the appropriate set(s) $S \subseteq A$ which maximizes the utility (value minus costs) that coalition v_S can achieve in the environment. The value and cost of the coalition are generic terms, which may in fact be functions of other domain-dependent and independent characteristics of the structure.

B.3.2. FORMATION

The complexity of the coalition formation task depends on the conditions under which the coalitions will exist, and the types of coalitions which are permitted. As with all organizations, operating in dynamic environments will be harder to maintain than in static ones. Additional complexity is also incurred if the partitioning of agents is not disjoint; that is, agents can have concurrent membership in more than one coalition. Uncertain rewards, self-interested agents and a potential lack of trust while coordinating add further obstacles to the process.

Sandholm [Sandholm *et al.*, 1999] analyzes the worst case performance of forming exhaustive, disjoint coalitions over a static agent population from a centralized perspective. They show that by searching only the two lowest levels of a complete coalition structure graph, an *a*-approximate value solution can be found to the partitioning problem, where $a = |A|$. Although the search of 2^{a-1} possible allocations still grows exponentially with *a*, the fraction of coalition structure needing to be searched approaches zero. They also present an anytime algorithm which can meet tighter bounds given additional time. Later work empirically evaluates the average-case performance of three anytime search techniques [Larson and Sandholm, 2000]. The algorithms' performances varied by domain characteristics; and no single technique were best in all conditions.

Shehory [Shehory and Kraus, 1998] has studied how coalitions may be used to enable task achievement by a group of agents. In their scenario, a set of interdependent (precedence) tasks must be accomplished, some of which require multiple agents to perform. The agents are cooperative and potentially heterogeneous in their capabilities. The strategy they employ draws on techniques used by Chvatal's greedy set covering algorithm [Chvatal, 1979], which tries to find the minimum set of subsets that together contain each member of a target set.

The initial values of all possible size-bounded coalitions are first computed and then iteratively refined in a distributed manner by the agents, taking into account task ordering and capability requirements. Once computed, the highest valued coalitions either disjoint or overlapping depending on the selection algorithm, are instantiated. This algorithm was also augmented to support dynamically arriving tasks. A drawback to this addition is that, in the worst case, the organization process needs to be redone for each task, incurring a significant communication cost. Also limiting the potential scalability of this approach is the need for each agent to have full knowledge

of the available agents and tasks.

Lerman [Lerman and Shehory, 2000] presents a scalable strategy where coalitions are formed between self-interested agents based only on local decision making. In this work agents operate in an electronic marketplace consisting of a number of extant purchase orders, with the objective of forming or joining a coalition of buyers that satisfied a need at the lowest price. Coalitions form around purchase orders, where agents form or join a coalition by adding a purchase request to an order, and can leave that coalition by removing their request. Agents in the system can move at will between purchase orders, searching for the one which offers the best value (lowest cost). An analysis based on differential equations shows that this strategy reaches equilibrium (later work [Lerman and Galstyan, 2001] expands on these mathematical techniques to analyze other distributed behaviours). It also has low communication and computational requirements. However, it does not provide guarantees on the achievable value or convergence rate, which would be affected by scale, and does not have a notion of deadlines on the purchase orders.

Soh [Soh, 2003] presents a technique where coalitions are dynamically created in response to the recognition of tracking tasks in a distributed sensor network. In this work, agents are assumed to have incomplete, uncertain knowledge and must respond to events in real time for goal achievement to be possible. As such, coalitions are formed in a satisfying, rather than optimal manner. An agent initiates coalition formation by first using local knowledge to select a subset of candidate partners that it believes will satisfy its requirements, both in terms of capabilities and willingness to cooperate. Next, it sequentially engages these candidates, in utility-ranked order, in argumentative negotiation, where offers and counteroffers are exchanged. This proceeds until satisfactory membership is decided, or the candidate list is exhausted.

Agents are cooperative, so during this negotiation process agents explicitly decide what coalition(s) they are willing to join based on perceived gains in utility. This approach does not make any guarantees about coalition value, or even that a satisfactory coalition will be found, but given the relatively short time in which an allocation must be made it would seem to be a reasonable strategy. In addition, reinforcement learning is used over the course of events to estimate candidate utility more accurately and select the most beneficial negotiation strategy, which should improve coalition value in the long run for reasonably stable environments. By storing preferences over multiple episodes, this learning also implicitly adds longevity to coalitions, giving organizational structures produced by this technique an interesting mix of dynamic and long-term characteristics.

B.4. TEAMS

An agent team consists of a number of cooperative agents which have agreed to work together toward a common goal [Fox, 1981, Tambe, 1997, Beavers and Hexmoor, 2001]. In comparison to coalitions, teams attempt to maximize the utility of the team (goal) itself, rather than that of the individual members. Agents are expected to coordinate in some fashion such that their individual actions are consistent with and supportive of the team's goal.

Within a team, the type and pattern of interactions can be quite arbitrary, as seen in *figure 39*, but in general each agent will take on one or more roles needed to address the subtasks required by the team goal. Those roles may change over time in response to planned or unplanned events, while the high-level goal itself usually remains relatively consistent (although exception handling may promote the execution of previously dormant subtasks).

This description of agent teams is quite general, and nearly any cooperative agent system has characteristics that are similar to these, if only

implicitly. However, systems that maintain an explicit representation of their teamwork or joint mental state are differentiated in their ability to reason more precisely about the consequences of their teamwork decisions [Jennings, 1995, Grosz and Kraus, 1996, Tambe, 1997]. For example, they will typically have representations of shared goals, mutual beliefs and team-level plans.

This type of representation provides flexibility and robustness by allowing the agents to explicitly reason about team-level behaviours, where a less explicit system may rely on a set of assumptions that ultimately make the system brittle in the face of unexpected situations.

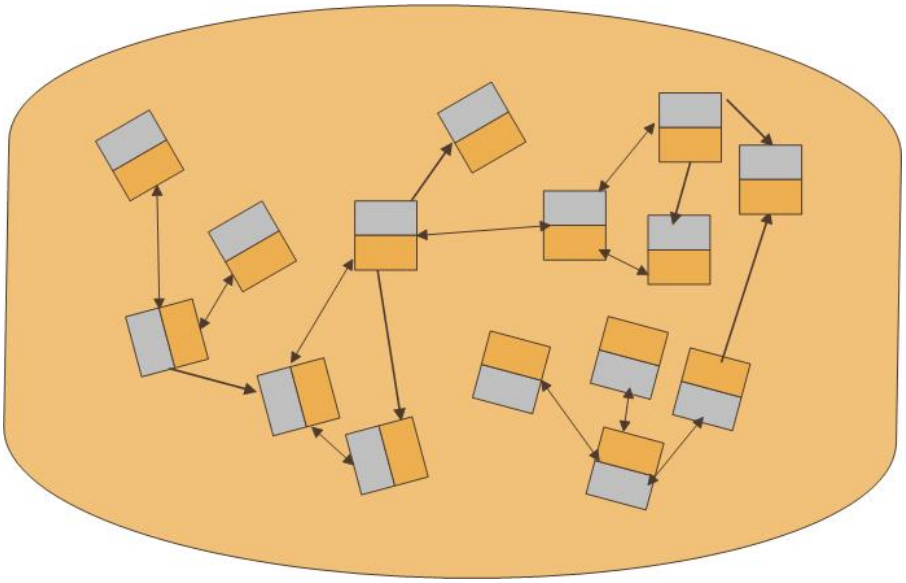


Figure 39. *Team-based organization.*

B.4.1. CHARACTERISTICS

The primary benefit of teamwork is that by acting in concert, the group of agents can address larger problems than any individual is capable of [Grosz and Sidner, 1990]. Other potential benefits, such as redundancy, the ability to meet global constraints, and economies of scale can also be realized

[Hexmoor and Beavers, 2001]. However, it is the ability of the team (members) to reason explicitly about the ramifications of inter-agent interactions which gives the team the needed flexibility to work in uncertain environments under unforeseen conditions.

The drawback to this tighter coupling is increased communication [Parker, 1993], so the team and joint goal representations, domain characteristics and task requirements are frequently used to determine what level of cooperation (and therefore communication) is needed [Pynadath and Tambe, 2002].

Jennings [Jennings, 1995] describes an electricity transportation management system which employs team-work to organize the activities of diagnostic agents. Lacking such structure, the agents were prone to incoherent and wasteful activities, since they did not always share useful behaviour information or propagate important environmental knowledge. By providing agents with an explicit representation of shared tasks and the means by which cooperation should progress, the agents were able to accurately reason about and resolve these interactions by employing team-level knowledge. Similarly, in [Tambe, 1997], teamwork is used to provide the structure and coordination needed by agents to address interdependent goals in dynamic environments, such as tactical military exercises and competitive soccer games. These works demonstrate how pathological, but hard to predict failures can be addressed if the plans are backed up by a general model of teamwork.

B.4.2. FORMATION

The challenges associated with team formation involve three principal problems: determining how agents will be allocated to address the high-level problem, maintaining consistency among those agents during execution, and revising the team as the environment or agent population changes [Jennings, 1995, Marsella *et al.*, 2001, Tidhar *et al.*, 1998].

The selection and role-assignment of agents that will work on the high-level problem depends on the goal's requirements, the capabilities of the candidate agents, and the knowledge of the selecting process itself [Tidhar *et al.*, 1996, Beavers and Hexmoor, 2001]. Initially, the process or agent performing the team construction must be aware of the agents which could potentially form the team. In the case of a static, reasonably sized agent population this can be done off-line as part of the system design or the members can be dynamically discovered and assessed. This latter technique can be accomplished using well-known discovery mechanisms such as the contract net protocol [Smith, 1980] or matchmaker intermediaries [Sycara *et al.*, 1997]. Once a suitable pool has been found, the capabilities and pre-existing responsibility of those agents must be evaluated relative to the needs of the goal.

Typically, agents are each denoted to have a set of capabilities, while the goal's subtask(s) are of a particular type. If an agent's capabilities include that sub-task's type, it can perform the task [Tidhar *et al.*, 1996, Fatima and Wooldridge, 2001]. The discovery mechanisms may include an implicit ranking technique, such as the bidding process employed in contract net, which makes the selection process relatively straightforward. Tidhar [Tidhar *et al.*, 1996] suggests a different technique where the agent characteristics are derived at compile time, either through designer input or automatic analysis of the agent's plan library. Candidate teams comprised of a sub-set of those agents may also be specified, which also are marked with their characteristics. At runtime, these characteristics are matched with the goal requirements as part of the team allocation search. By including these characteristic labels, the number of possible team combinations can be greatly reduced.

Tambe's STEAM [Tambe, 1997] architecture provides a flexible method for representing and adapting team behaviours. It is based on the joint intentions frame-work [Levesque *et al.*, 1990], which formally defines how

agents should reason over joint commitments and shared goals, and SharedPlans theory [Grosz and Kraus, 1996], which provides a formal way to encode and reason about joint plans, intentions and beliefs. Together, these help ensure a consistency of belief, or a desire to enact such a belief, across all team members.

The commitments formed through the joint intentions process provide the explicit structure needed to reason about and monitor performance on a team level. Team plans are represented using a hierarchical decomposition tree, with nodes representing tasks for both teams and individuals, with associated preconditions, application and termination rules. Agents may simultaneously take part in several different tasks, and corresponding roles.

The team's cohesion is derived primarily from the joint intentions created as part of executing the team plans. Upon selecting a team task, agents first broadcast this intention to affected agents, and wait until a commitment to that task has been established between all participants. The existence of this commitment directs agents to propagate changes whenever the task is perceived to be achieved, unachievable or irrelevant, before taking local action itself.

This trades off the potential reaction speed of the team and the cost of communication with group conformity. A decision theoretic approach is used to guide communication acts, which explicitly trades off the costs of communication with those of inconsistent beliefs. Nair [Nair *et al.*, 2003b] has also explored the possibility of using simulated emotions to provide the motivation to enforce team-level behaviours.

In STEAM, monitoring and repair of the team is accomplished with the use of role constraints [Tambe, 1997]. Team members are assigned a role, based on the particular task they are working on. These roles are further constrained such that some particular combination of them (e.g. and, or) are needed to accomplish the task. One can then monitor if a task is achievable by

monitoring the health of the individual agents, and using that information to evaluate if the role constraints are satisfied. Such monitoring can be performed through explicit queries, environmental observations or by eavesdropping on communication, which can reduce the increased communication usually associated with teams.

Kaminka [Kaminka *et al.*, 2002] has demonstrated that the latter technique can perform well when coupled with a plan-recognition algorithm. Failures can thus be detected, and potentially resolved through an appropriate role-substitution, or the task abandoned if no substitution is possible. Alternately, one could use a diagnosis system [Jennings, 1995, Horling *et al.*, 2001] to more precisely identify the root cause of the failure. Interestingly, this repair operation can itself be cast as a team task, so mutual agreement that a repair is necessary must be achieved before potentially drastic measures are taken.

Nair [Nair *et al.*, 2003a] shows how an MDP incorporating team and role-allocation knowledge can improve the system's response in cases of multiple role failure. In this case, a suitable locally optimal policy for the reallocation decision problem can be found by analyzing the team's plans, and then used to guide on-line responses to failures. This work showed that such policies can provide improved performance versus more heuristic and analytic techniques. A similar technique was also shown in that work to improve initial role allocation.

Tidhar [Tidhar *et al.*, 1998] uses a similar hierarchical plan representation to represent teamwork in a tactical air mission scenario. Team membership and role assignment are performed by matching agent capabilities to one or more role's requirements. As in STEAM, teams can be broken down into sub-teams, and agents may use both implicit (observation) and explicit (messaging) forms of coordination.

The Generalized Partial Global Planning (GPGP) framework also employs techniques that allow agents to act using team semantics [Decker and Lesser, 1992, Lesser *et al.*, 2004]. Where a STEAM-driven system will typically organize in an explicit, controlled fashion in response to a perceived goal, a GPGP-team is created in a more dynamic, emergent fashion. GPGP agents are provided with a set of individual plans which model a range of alternative ways that goals may be achieved. The sub-goals modelled in these plans may affect or be affected by other agents in the environment, although this may not be initially recognized.

By communicating with one another and exchanging plans and schedules, these *non-local interrelationships* between tasks may be recognized. For example, the results from one agent's activity may be a strict prerequisite for another agent's task. They may alternately be a facilitating, but not required input to a task. By recognizing these interrelationships, and sharing knowledge of what goals are being pursued, agents gradually build an internal model of how their actions may affect others. This knowledge is similar to that created by the more formal joint intentions of STEAM, and allows agents to influence local behaviour and communicate results as if they were members of a common team.

B.5. CONGREGATIONS

Similar to coalitions and teams, agent congregations are groups of individuals who have banded together into a typically flat organization in order to derive additional benefits. Unlike these other paradigms, congregations are assumed to be long-lived and are not formed with a single specific goal in mind. Instead, congregations are formed among agents with similar or complementary characteristics to facilitate the process of finding suitable collaborators, as modelled in *figure 39*. The different shadings in this figure represent the potentially heterogeneous purpose behind each grouping,

in comparison to the typically more homogeneous coalitions in *figure 40*. Individual agents do not necessarily have a single or fixed goal, but do have a stable set of capabilities or requirements which motivate the need to congregate [Brooks *et al.*, 2000, Griffiths, 2003]. Analogous human structures include clubs, support groups, secretarial pools, academic departments and religious groups, from which the name is derived.

Congregating agents are expected to be individually rational, by maximizing their local long-term utility. Group or global rewards are not used in this formalism [Brooks *et al.*, 2000]. It is this desire to increase local utility which drives congregation selection, because it is the utility that can be provided by a congregation's (potential) members that determine how useful it is to the agent.

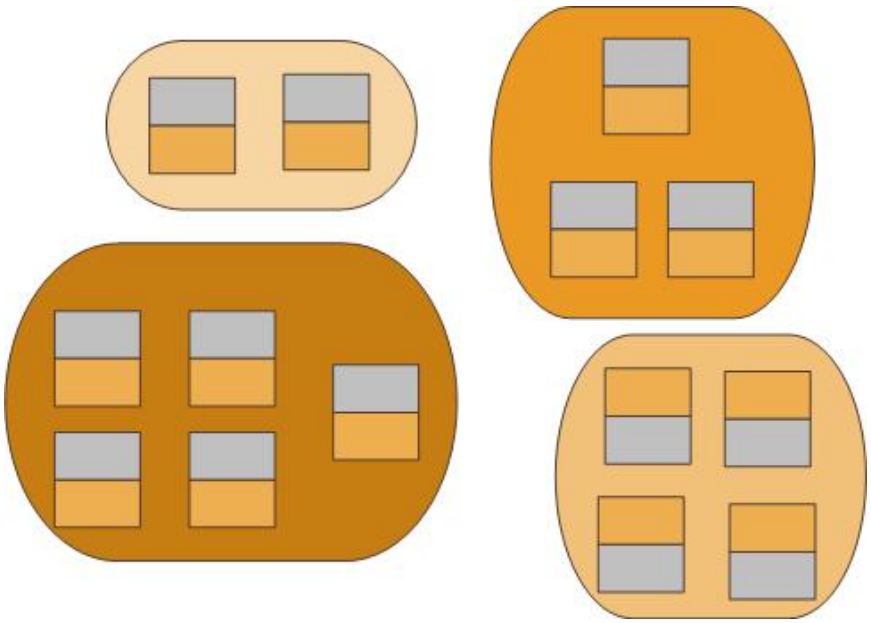


Figure 40. *Congregations of agents.*

Agents may come and go dynamically over the existence of the congregation, although clearly there must be a relatively stable number of participants for it to be useful. Agents must also take enough advantage of the congregation so that that the time and energy invested in forming and finding the group is outweighed by the benefits derived from it. Since congregations are formed in large part to reduce the complexity of search and limit interactions, communication does not occur between agents in different congregations, although the groups are not necessarily disjoint (i.e., an agent can be a member of multiple congregations).

The net result of the congregating behaviour is an arrangement that can produce greater average utility per cycle spent computing or communicating [Brooks and Durfee, 2002].

B.5.1. CHARACTERISTICS

Although congregations can theoretically share many of the same benefits of coalitions, their function in current research has been to facilitate the discovery of agent partners by restricting the size of the population that must be searched. As a secondary effect these groupings can also increase utility or reliability by creating tighter couplings between agents in the same congregation, typically by imposing higher penalties for decommitment or increasing information sharing among congregating peers. The downside to this strategy is that the limited set may be overly restrictive, and not contain the optimal agents one might interact with given infinite resources. So, in forming the congregation, one is trading off quality and flexibility for a reduction in time, complexity or cost. If an appropriate balance can be found, this will result in a net gain in utility.

This hypothesis is borne out in the experiments from an information economy domain [Brooks and Durfee, 2002]. This work varied the number of congregations that agents were allowed to form. Since the population size was

static, the average congregation size decreased as the number of congregations increased. The accumulated quality decreased proportionally because of less flexibility in agent interactions. However, these smaller congregations also incurred lower overhead, and thus had less cost. A median point was discovered in the space which produced maximum value.

B.5.2. FORMATION

Like coalition formation, congregation formation involves selecting or creating an appropriate group to join, and suffers from similar complexity problems as the agent population grows. Because congregations are more ideologically or capability driven, and there is usually no specific goal or task to unite them, one must first define how these groups may be differentiated. In [Brooks and Durfee, 2003] Brooks proposes using labels to address this problem. A label is a suitably descriptive tag assigned to each congregation which serves to both distinguish it from other groups and advertise the characteristics of its (desired) members. Assuming that agents have an ordered preference for such labels, the congregators' action is simply to move to the congregation for which it has the highest preference.

The problem is then to create a number of logical points where agents may congregate and then decide upon the labels each congregation point will have; these labels help determine the makeup of the population which gathers there. Each agent was placed into one of several affinity groups, and a congregation is stable if and only if it contains only members of the same affinity group.

Different numbers of labellers were then added which could attach labels to the congregation points. As with the congregators, the labellers were stable if and only if the congregation they provided the label to was homogeneous. The experimental and analytic results demonstrated that by increasing the number of labellers the system converged more quickly.

Brooks [Brooks and Durfee, 2002] presents a variation of this formation technique used in an information economy which also takes into account the costs associated with congregation size. In this scenario there are a set of buyers and sellers. Each buyer has an information preference, and each seller may choose what type of information to offer.

The buyer's preference is soft – they have an optimal type, but are also willing to purchase related information, where similarity determines how much they are willing to pay. Instead of explicitly labelling congregation points, agents freely move through the system seeking groups that provide acceptable utility. The scenario is episodic, where during each episode agents elect to stay in place or randomly move to a new congregation. At the end of each episode an auction takes place from which buyers and sellers obtain their utility. The utility is based on the price of the goods bought and sold, combined with the costs incurred during the auction. This cost, divided uniformly among the congregation members, is proportional to the complexity of the auction, which is itself determined by the number of participants. Satisfied agents remain, while those which do not obtain enough utility moves. This process results in an emergent population of congregations that trades off utility for computation time.

Griffiths' notion of a clan closely parallels the definition of a congregation [Griffiths, 2003]. He presents a technique where clans are formed as part of a self-interested activity to increase local utility or decrease the probability of failure. If a motivating factor is exhibited by the agent, such as a desire to increase information gain or decrease commitment failure, clan formation may be initiated. Clan formation begins with the agent identifying how large a clan it wishes to create, which is based on the competing utility (in value added) and cost (in computational complexity) that grow in proportion to clan size. A trust value is then used to determine what agents it could invite, while the perceived capabilities or benefits of those individual agents are used

to determine the appropriately sized subset that it will invite. In lieu of a negotiation process or explicit reward, invitation recipients determine if they will accept the invitation based first on their trust in the sender, and second on the perceived local gain they would receive by joining. The sender includes information about itself in the invitation as a sort of capability advertisement to facilitate this determination. If a sufficient number of agents agree, the clan is formed, otherwise the attempt is abandoned.

Although it does not strictly deal with congregating agents, Sen's work on reciprocal behaviour [Sen, 1996] has some of the same characteristics. In this system, agents become more inclined to cooperate or assist another agent when it has a favourable history with that other agent. Specifically, agents track if others have cooperated with it in the past, or if it has cooperated with them, along with the approximate costs of those experiences.

If an agent has a favourable balance of cooperation, it will be more inclined to give or receive assistance. The cooperation decision process is stochastic, enabling reciprocal relationships to be created or promoted even when a strictly positive balance does not exist. Weak groups may form between agents using this strategy who have complementary capabilities, which is similar to the notion of congregations presented here.

Because agents will more likely communicate with those that will help it, interactions can become implicitly confined within the group. These groupings are not formalized or well-defined, however, and communication is not necessarily restricted by the approximate boundaries that form. Sen showed that, among a group of self-interested agents operating in a package delivery domain, a population containing reciprocal agents outperformed a selfish population.

B.6. SOCIETIES

Drawing from the experiences with biological societies, a society of agents intuitively brings to mind a long-lived, social construct. Unlike some other organizational paradigms, agent societies are inherently open systems. Agents of different stripes may come and go at will while the society persists, acting as an environment through which the participants meet and interact. A canonical example of this paradigm is the electronic marketplace, consisting of buyers and sellers striving to maximize their individual utility [Wellman and Wurman, 1998, Artikis, 2003]. A more ambitious example is the "agent world", a permanent operating environment or agents [Dellarocas and Klein, 2000a, Willmott *et al.*, 2001]. Agents will have different goals, varied levels of rationality, and heterogeneous capabilities; the societal construct provides a common domain through which they can act and communicate. Societies are also more ephemeral constructs than others paradigms explained so far. They impose structure and order, but the specific arrangement of interactions can be quite flexible. Within the society, agents may be sub-organized into other organizations, or be completely unrelated.

A second distinguishing characteristic of societies is the set of constraints they impose on the behaviour of the agents, commonly known as social laws, norms or conventions. This arrangement is shown abstractly in *figure 41*, where the agents within the society have been provided with a set of specified norms. These are rules or guide-lines by which agents must act, which provides a level of consistency of behaviour and interface intended to facilitate coexistence. For example, it might constrain the type of protocol(s) agents can use to communicate, specify a currency by which they can transfer utility, or limit the behaviours the agent can exhibit in the environment. Penalties or sanctions may also exist to enforce these laws.

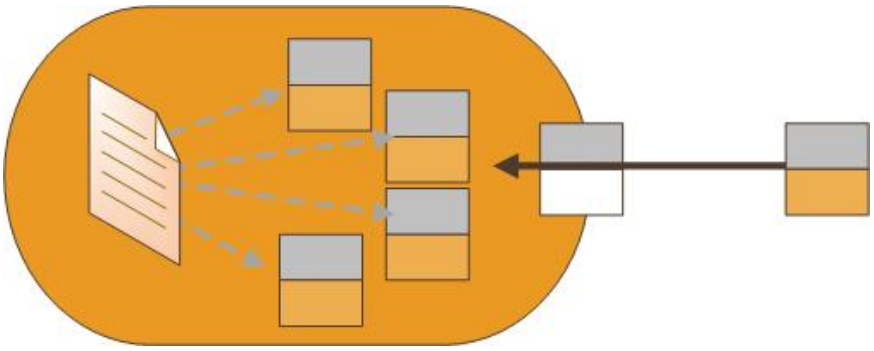


Figure 41. *An agent society.*

The set of laws embedded in a society must strike a balance among objectives [Fitoussi and Tennenholtz, 2000]. It must be sufficiently flexible that goals are achievable, but not so much so that the beneficial constraints provided by the laws are lost. It must also be fair, such that the goals of one class of individuals are not incorrectly valued higher than those of another. These issues arise naturally in any structured, multiple participant system; Moses argues that most multi-agent systems have some form of social laws in place, if only implicitly [Moses and Tennenholtz, 1995].

B.6.1. CHARACTERISTICS

In [Shoham and Tennenholtz, 1995], Shoham presents a grid world where robots must move from one location to another in accordance with a set of dynamically arriving tasks. Conflicts can arise when two or more agents attempt to occupy the same location at the same time along their chosen paths. They argue that a centralized solution is untenable, because of the potentially large number of interactions that must be continuously reasoned over in the heterogeneous population. Neither is a fully decentralized solution appropriate, because of the number of negotiation events that would need to take place at each time step. This motivates the need for *"traffic laws"*, a type of social law which does not eliminate such interactions, but should minimize

the need for them. The traffic laws in this research are computed offline, and constrain the robots' movement patterns in such a way that collisions do not occur and destinations are reachable within a bounded amount of time. Vehicular traffic laws serve the same purpose in human societies. When driving a car there is no central authority which determines when and where to go, and neither is there a free-for-all on the roads where one must talk to every other driver before proceeding. The challenge then is to design a set of laws that minimizes conflicts and encourages efficient solutions.

Although social laws were used to provide efficiency benefits in the work above, the purpose of an agent society is not always as quantitatively-driven as other organizational constructs. Indeed, most research on agent societies is more concerned with how the concepts they embody can be used to facilitate the construction of large-scale, open agent systems in general. For example, Moses [Moses and Tennenholtz, 1995] argues that social laws can provide a formal structure upon which more complex inter-agent behaviours can be built. By limiting and enforcing these restrictions, agents can make simplifying assumptions about the behaviour of other agents, which can make interaction and coordination more tractable.

In additional to formalizing normative behaviours, mechanisms may also be established to ensure or encourage that such laws are respected. One approach accomplishes this through explicit representations of reputation or trust [Mui *et al.*, 2002, Ramchurn *et al.*, 2005, Sabater and Sierra, 2001]. An agent's behaviour and interactions are observed by its peers and evaluated in the context of the norms it has agreed to. Deviation from those norms will result in a worsening reputation. This decreased reputation can in turn affect the utility the agent obtains, through increased decommitment penalties or competition from more reputable peers. In a rational agent this will serve as a deterrent to violating conventions.

A different, but complementary approach instantiates and enforces social laws using social institutions provided in the environment [Dellarocas and Klein, 2000a, Colombetti *et al.*, 2004]. Agents are expected to formalize their interactions using contracts, which are independently verified by these institutions, thereby relocating some of the traditionally agent-centric complexity into a service available to the population as a whole. This reduces the burden placed on agent designers, and provides a mechanism where systemic (non-localized or long-term) failures may be detected more readily. This more rigorous enforcement of social laws also helps address the problem of unreliable, dishonest or malicious agents operating in the open environment.

Huhns [Huhns and Stephens, 1999] provides similar motivation for common communication languages, shared or interoperable ontologies and coordination and negotiation protocols, all of which may be specified as part of the society's structure. These beliefs can be supported by the experiences acquired in real life. It should be clear that complex human societies are founded upon the ability to interact with one another. Mutually understood and respected norms simplify many aspects of day-to-day existence. These principles can be used to the same effect in agent societies.

B.6.2. FORMATION

There are two aspects to the society formation problem. The first is to define the roles, protocols and social laws which form the foundation of the society. Given such a definition, the second problem is to implement the more literal formation of the society, by determining how agents may join and leave the defined formation.

If the society is to be an open and flexible system, its structure must be formally encoded so that potential members may analyze it and determine compatibility. This description can be as simple as a set of common interfaces

that must be implemented, or a complex description of permissible roles, high-level objectives and social laws. Dignum [Dignum, 2004, DignumMeyer *et al.*, 2002] presents a three-part framework, consisting of organizational, social and interaction models. The organizational model defines the roles, norms, interactions and communication frameworks that are available in the environment. The social model, instantiated at run-time, defines which roles agents have taken on. The interaction model, also created at run-time, encodes the interactions between agents that have been agreed-upon, including the potential reward and penalties. The latter two models are supported by contracts between the relevant entities. This formalism is similar to that proposed by Artikis [Artikis, 2003], which provides additional details describing operators that can be used to encode social laws, roles and normative relations. Because the society is intended to be open, these structures do not involve the internal implementation of agents, but describe only the intended or expected externally observable characteristics of the participants and environment.

Assuming it is possible to encode the social laws in a way that makes them intelligible to agents, one still faces the challenge of determining what conventions should be enacted. Fitoussi [Fitoussi and Tennenholtz, 2000] presents a notion of minimal social laws, where he argues that one should choose the smallest and simplest set of norms that address the needs of the society. This is consistent with the trade-off between flexibility and complexity mentioned above. Work has also been done exploring the dynamic emergence of norms, for when social laws cannot be specified off-line or if there is a desire for the corpus to be responsive to changing conditions [Axelrod, 1986, Hewitt, 1986]. Walker and Wooldridge [Walker and Wooldridge, 1995] propose and evaluate a number of ways that a group of agents can reach norm consensus based on locally available information.

Dellarocas defines the act of an agent entering a society to be the socialization process [Dellarocas and Klein, 2000a]. In that work, they suggest this can be accomplished through an explicit negotiation process between the agent and a representative of the society, as shown in the left side of *figure 40*. This exchange results in a social contract, or an explicit agreement made between the agent and the society indicating the conditions under which the agent may join that society. This allows the possibility of capable agents dynamically learning, and potentially negotiating over, the rules it must abide by in that society. This naturally extends to multi-society environments, where an agent's skills and goals define how good a fit it is with a particular society. Some of the challenges associated with operating in multi-society environments seem to be comparable, though larger in scale, to those encountered during coalition or congregation formation.

Because of their inherent flexibility, a great deal of additional complexity may be associated with social organizations. Sophisticated legal systems, communication bridges, ontologies, exception handling services, directories may all be part of the society model [Dellarocas and Klein, 2000a, Dignum, 2004, Klein *et al.*, 2003]. Some or all of these may be directly instantiated by trusted agents taking on so-called facilitation roles (differentiated from the operational roles taken on by worker agents). Of course, agents acting in the society must have a certain level of sophistication to know how and when to use such services. An interesting almost-paradox exists in this relationship. Although the society exists in part to reduce the complexity burden imposed on the participants, the participants must raise their level of complexity to take advantage of these benefits. In the case where interactions with some or all social services are mandatory (e.g. legal or arbitration services), this additional complexity is similarly unavoidable and can act as a barrier to entry.

B.7. FEDERATIONS

Agent *federations*, or *federated systems*, come in many different varieties. All share the common characteristic of a group of agents which have ceded some amount of autonomy to a single delegate which represents the group [Wiederhold, 1992, Genesereth, 1997]. This organizational style is modelled on the governmental system of the same name, where regional provinces retain some amount of local autonomy while operating under a single central government. The delegate is a distinguished agent member of the group, sometimes called a facilitator, mediator or broker [Sycara *et al.*, 1997, Hayden *et al.*, 1999].

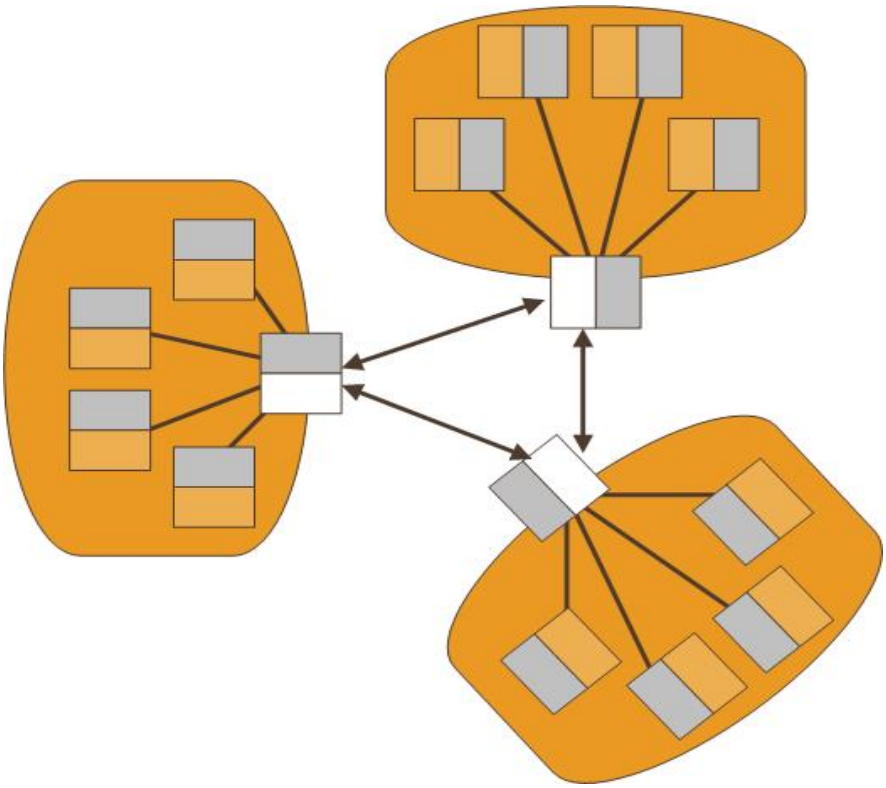


Figure 42. *An agent federation.*

Group members interact only with this agent, which acts as an intermediary between the group and the outside world, as shown in *figure 42*. In that figure each grouping is a federate, and the white agent situated at the edge of each federate is the delegated intermediary. Typically, the intermediate accepts skill and need descriptions from the local agents, which it uses to match with requests from intermediaries representing other groups. In this way the group is provided with a single, consistent interface. This level of indirection is similar to that seen in holons, and provides some of the same benefits.

B.7.1. CHARACTERISTICS

The capabilities provided by the intermediary are what differentiate a federation from other organizational types. The intermediary functions on one hand by receiving potentially undirected messages from its group members. These may include skill descriptions, task requirements, status information, application-level data and the like. These will typically be communicated using some general, declarative communication language which the facilitator understands [Genesereth, 1997]. Outside of the group, the intermediary sends and receives information with the intermediaries of other groups. This could include task requests, capability notifications and application-level data routed as part of a previously created commitment. Implicit in this arrangement is that, while the intermediary must be able to interact with both its local federation members and with other intermediaries, individual normal agents do not require a common language as they never directly interact. This makes this arrangement particularly useful for integrating legacy or an otherwise heterogeneous group of agents [Genesereth, 1997, Shen and Norrie, 1998].

The intermediary itself can function in many different capacities. It may act as a translator, perform task allocation, or monitor progress, among other things. An intermediary which accepts task requests and allocates those

tasks among its members is known as a broker or a facilitator. As part of the allocation, the broker may decompose the problem into more manageable subtasks. This allows agents to take advantage of all the capabilities of the (potentially changing) federation, without requiring knowledge of which agents perform a task or how they go about doing it. This reduces the complexity and messaging burden of the client, but also has the potential of making the broker itself a bottleneck [Hayden *et al.*, 1999] (a possibility common to all intermediaries).

An intermediary acting as go-between among agents is known variously as a translator, embassy or mediator depending on its specific characteristics. Embassy agents provide a layer of security for members of their federation, by having the ability to deny communication requests. Mediator agents store representations of all related parties, reducing their individual complexity by providing a layer of abstraction. This capacity can be further exploited to arbitrate conflicts [Mailler and Lesser, 2004]. Intermediaries which provide the ability to track the state of one or more of its participants are known as monitors. For example, result information can be automatically propagated to interested parties. Of course, one or more of these roles may be combined into a single intermediary which offers several types of services.

B.7.2. FORMATION

Genesereth [Genesereth, 1997] describes how a general federated system would work. All agents are expected to communicate using an Agent Communication Language (or ACL, a somewhat-generic term used by many researchers to describe their agents' communication protocol), which in this work is a combination of the first-order predicate calculus KIF with the KQML agent messaging language. Knowledge and statements sent between agents are encoded as KIF statements, which are then wrapped in KQML to

provide a standard mechanism for specifying the sender, receiver, intent, and so forth. This provides a common language and set of behavioural constraints that will allow the various agents to interact. Not all agents must implement the entire class of concepts in the ACL, but the aspects they do use must be correct with respect to the ACL's specification.

In addition, although they speak the same language, not all agents must use the same vocabulary to describe a particular situation, although to interact there must be an intermediary capable of translating the vocabularies. The system is initialized with a set of intermediaries called facilitators, which serve many of the roles outlined above, notably brokering. Agents connecting to the system start by sending their capabilities to the local facilitator. Implicit in this communication is the notion that the agent is willing to use those capabilities in service of requests posed by the facilitator. Needs are similarly routed to the facilitator, which then attempts to find other facilitators that can service that need. Each facilitator provides a "yellow pages" function which supports this search. Khedro's Facilitators [Khedro and Genesereth, 1995] and the jointly developed PACT project [Cutkosky *et al.*, 1993] have produced very similar systems that also use a common ACL and a community of intermediaries to produce a robust and dynamic task decomposition and allocation scheme among a group of heterogeneous participants.

The MetaMorph I [Maturana *et al.*, 1999] and II [Shen and Norrie, 1998] architectures described by Maturana and Shen demonstrate a federated agent system for use in intelligent manufacturing. In this domain, agents are used to drive aspects of product design and manufacturing, contending with heterogeneous resources, dynamically changing conditions, and hard and soft constraints on behaviour. MetaMorph's name is derived from the fact that the system can continuously change shape, adapting to new conditions as they are perceived. This is accomplished in part through the use of intermediaries called mediators, which are responsible for brokering, recruiting and conflict

resolution services. The recruiting service is similar to brokering, but is differentiated by the fact that the intermediary can remove itself from the relationship once the partners have been discovered. This weaker form of federation provides efficiency gains at the cost of less flexibility, both due to the loss of the layer of abstraction that exists in the brokered approach. The federations themselves are dynamically created in response to new task arrivals or requests from other groups using a contract net [Smith, 1980] approach, or are statically created from agents in a common subsystem (e.g. tools, workers, etc.).

B.8. MARKETS

In a *market*-based organization, or *marketplace* as shown in *figure 43*, buying agents (shown in white) may request or place bids for a common set of items, such as shared resources, tasks, services or goods. Agents may also supply items to the market to be sold. Sellers (shown with a darker lower part), or sometimes designated third parties called auctioneers, are responsible for processing bids and determining the winner.

This arrangement creates a producer-consumer system that can closely model and greatly facilitate real-world market economies [Wellman, 2004]. These latter systems fall into the more general category of agent-mediated electronic commerce [Guttman *et al.*, 2001]. Because of this similarity, a wealth of research results from human economics and business can be brought to bear on agent-based markets, creating a solid theoretical and practical foundation for creating such organizations [Wellman, 1993, Wellman and Wurman, 1998, Corkill and Lander, 1998].

Markets are similar to federated systems in that a distinguished individual or group of individuals is responsible for coordinating the activities of a number of other participants. Unlike a federation, market participants are typically competitive. In addition, participants do not cede operational

authority to those distinguished individuals, although they do trust the entities managing the market and abide by decisions they make. It is also common for markets to operate as open systems [Wellman, 2004], allowing any agent to take part so long as it respects the system's specified rules and interface. As such, they share some of the benefits and drawbacks of societies.

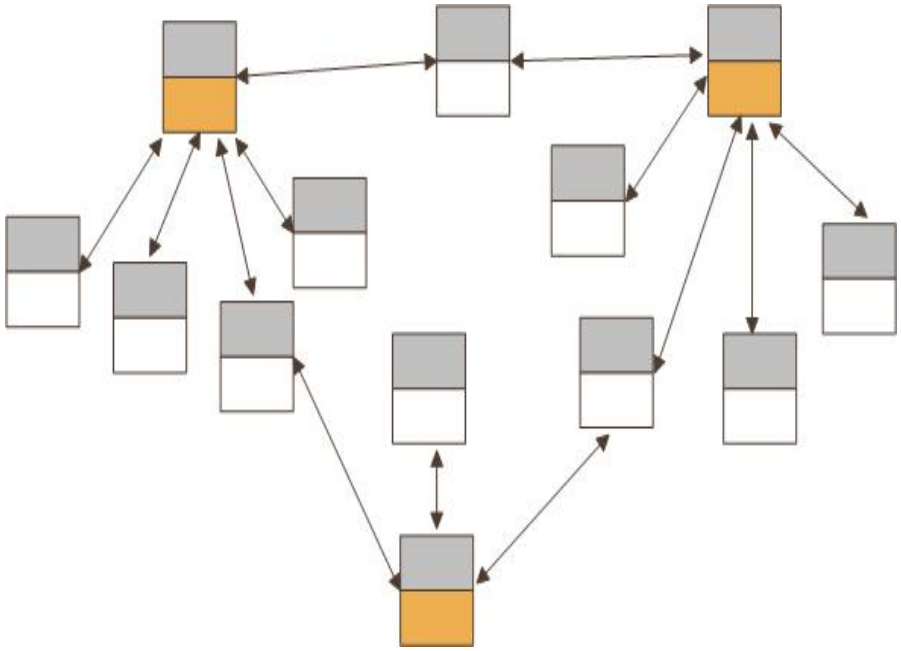


Figure 43. *A multi-agent marketplace.*

When using the terms *"buyer"* and *"seller"*, one may implicitly assume that an artefact will eventually be transferred in exchange for some form of compensation [Chavez and Maes, 1996, Tsvetovatyy *et al.*, 1997]. Although this paradigm is common, it is not always the case, and market-based organizations have been used in various projects to accomplish less obvious goals. For example, Wellman [Wellman *et al.*, 2001] proposes using a market-based approach to perform decentralized factory scheduling. In this work, each factory job is associated with a duration, deadline and value. The factory itself, acting as the seller, has a reserve price associated with the time

slots it has available. Agents bid on a set of slots that have sufficient total time to satisfy the job duration and do not exceed the deadline, using the job value as a maximum bid price. Market forces will cause agents to seek out the most cost-effective time slots, while higher-valued jobs will naturally take precedence over lower ones. This should lead to an efficient allocation of (time) resources, while maximizing the factory's overall usefulness.

Bussman [Bussmann and Schild, 2000] has developed an auction-based manufacturing control system with a similar purpose, where agents are used to represent workpieces, transportation conveyors and machines. In this work, machines bid for the right to work on workpieces, which act as sellers, by relating an expected time to completion. When a machine's bid is accepted, a series of additional negotiations between the workpiece and the conveyors move the piece to the appropriate location. Yet another example is the Mariposa distributed database system [Stonebraker *et al.*, 1996], which uses market-based techniques to optimize query processing. Individual nodes buy and sell fragments of information. Queries inserted into the system are associated with a biding profile, indicating how much the user is willing to pay. A brokering process takes the query and requests bids from relevant nodes. Who then submit bids in an effort to win the rights to process the query
More generally, Wellman proposes the notion of market-oriented programming [Wellman, 1993], which uses the marketplace paradigm as a general programming methodology that can efficiently address multi-commodity flow and resource allocation problems. His WALRAS framework that implements this concept has been used to create solutions for transportation logistics, product design and distributed information services. Many other marketplace frameworks have also been developed for general use [Chavez and Maes, 1996, Rodríguez *et al.*, 1997, Collins *et al.*, 1998, Collis and Lee, 1999, Cuni *et al.*, 2004]; Kurbel and Loutchko provide a comparative analysis of structure and function [Kurbel and Loutchko, 2003].

B.8.1. CHARACTERISTICS

Markets excel at the processes of allocation and pricing [Wellman and Wurman, 1998]. If agents bid correctly (i.e. make truthful bids according to their perceived utility gain if they win), the centralized arbitration provided by the auctioneer can result in an effective allocation of goods. The Kasbah system [Chavez and Maes, 1996] is an example of an agent-based marketplace that demonstrates many of the typical characteristics of this type of organization. Agents in Kasbah are segregated into two categories: buyers and sellers. Both types indicate the type of object they are interested in (buying or selling) with a feature vector, along with a desired price, a threshold price (lower or upper bound), and a negotiation strategy that controls how their offered price changes over time. A sale occurs when a seller's price matches what a buyer is willing to pay. The objects being sold in this system represent the targets of the allocation process, and the price is determined dynamically according to supply and demand. The mechanism that is employed in Kasbah corresponds to an intuitively fair way to allocate among competitors, at least from a self-interested point of view: all agents gradually compromise, and the agent willing to meet the seller's price first wins.

The behaviours embodied in a marketplace, namely the existence of buyers and sellers, a potential multitude of goods, and competition among participants, make such organizations intrinsically linked with the properties of auctions. Kasbah is an example of a two-sided auction, because both sides compromise. If one of the two parties maintained a fixed price, it would be one-sided auction. Many other types of auctions exist to service the different needs of different communities, each with their own characteristics [Wurman *et al.*, 2001, Kurbel and Loutchko, 2003].

For example, in a combinatorial auction, participants bid on collections of goods, rather than single objects. In a reverse auction, sellers bid

rather than buyers. In sealed-bid auctions, the participants do not see competing bids while the auction is in progress. In continuous auctions, a pool of items exists, exchanges occur as soon as two compatible bids are made, and the bidding process continues uninterrupted. The particular type of auction which is employed dictates the manner in which the participants interact. Much of the complexity involved in designing an effective market and marketplace agent revolves around understanding the subtleties of the auction's characteristics, and crafting an appropriate strategy based on that knowledge.

There are two drawbacks to market-based organizations. The first is the potential complexity required to both reason about the bidding process and determine the auction's outcome. The former computation may require a detailed approximation of competitors' beliefs, a practice known as counterspeculation, especially in single-shot or sealed bid auctions [Tsvetovatyy *et al.*, 1997]. The latter computation, also known as clearing the trade, can be particularly difficult in the case of combinatorial auctions. This is known to be a NP-complete problem [Sandholm, 2002], although solutions have been devised that have good performance in practice [Sandholm, 2006]. The second is security; in addition to the practical network-related security issues inherent in any open system, one must also be able to verify the validity of the auction approach itself.

For example, the bidding strategy used in the Kasbah system is vulnerable to a form of cheating known as collusion. If two or more bidders in the system agree to reduce their rate of compromise, they have a chance to artificially lower the final sale price. It is also important that the bidding process does not reveal information about the participants. For example, if a seller could determine the threshold prices of some of its buyers, it could simply wait until the maximum such price is reached, thereby artificially increasing the sale price. Some of these issues can be resolved by selecting an

appropriate auction type. The Vickrey auction's structure [Vickrey, 1961], where the highest bidder wins but pays the second highest bid price, promotes truthful bidding and discourages counterspeculation. Enforcing anonymity and secure communication channels can also help avoid many common pitfalls.

B.8.2. FORMATION

As is the case of many open systems, marketplaces are frequently static, pre-existing entities that do not require a formal creation process beyond starting the actual market process (if any) and allowing agents to connect. The well-known Trading Agent Competition market [Wellman and Wurman, 1999] operates in such a fashion, albeit for a limited amount of time. They may have certain barriers to entry, such as respecting a defined programming interface, implementing a particular transaction language, and respecting the rules of the market's auction type. These entry conditions are similar to those discussed earlier in the context of societies, although there is generally no formal negotiation or socialization process involved. Wellman [Wellman, 2004] outlines a number of other practical characteristics that should be exhibited for a marketplace to be successful. They must maintain temporal integrity, meaning that the outcome of an auction depends on the arrival sequence of bids, and is independent of any delays internal to the market itself. Transactions performed by the market must be atomic, that is, they have no effect if they fail or are cancelled prior to completion. As noted above, they also require attention to security risks, so that participant information is adequately protected and the auction process itself is kept safe from conventional attacks, particularly if there is an actual exchange of goods, information or currency in the market. Markets may also incorporate product discovery services, banking services, brokering middle-agents and negotiation support, to reduce the burden placed on the participants [Tsvetovatyy *et al.*, 1997, Guttman *et al.*, 2001].

Other works have explored dynamic formation of markets. Brooks has used the notion of congregations to dynamically form markets within a group of agents [Brooks and Durfee, 2002]. Recall that congregations are groups of agents which have banded together because of some common long-term interest or goal. In this work, that long term goal is the cost-effective exchange of goods or services. In a large population, it can be difficult to directly find suitable trading partners, and expensive to contact or broadcast to all possible partners. A suitably formed congregation serves to limit the scope of this search or broadcast, which in turn facilitates the marketplace creation.

A relatively new concept being exploited in both human [Mowshowitz, 1997] and agent [Ahuja and Carley, 1999, Foster *et al.*, 2004, Cardoso and Oliveira, 2004] organization research is the virtual organization (VO). A virtual organization is one that has a fixed purpose (e.g., to provide a set of services) but a potentially transient shape and membership. The key characteristics of a VO are that they are formed by the grouping and collaboration of existing entities, and there is a separation between form and function that precludes the need to rigidly define how behaviour will take place. This provides flexibility in how a particular goal is satisfied, by allowing the system to adapt the set of participants to meet resource availability and service demand. The concept is similar to the coalition and congregation paradigms discussed earlier, and have many of the same benefits as a federation, although a virtual organization can generally be thought of as an entity in and of itself more so than an empty coalition or congregation.

The CONOISE project has explored the dynamic creation of virtual organizations within a larger marketplace environment [Norman *et al.*, 2004]. In this context, the creation of a VO can be thought of as the creation of a new market entity (buyer or seller) from a group of existing participants. This can give those participants greater leverage, efficiency or reliability as they combine their producing or consuming power. The members of a VO may

remain distinct when outside of the marketplace, but within the market they act as a single unit. For example, two producers might combine to offer a new joint product. Two consumers might combine to obtain greater buying power. In responding to bids, a VO will then be able to offer the union of services or goods over all its members. VOs may also split when the relationship is no longer beneficial or if levels of trust or reputation have been sufficiently degraded. In all cases, the shape of the market is affected as these changes are made, and thus the market as a whole will evolve over time based on the needs and capabilities of the participants, and the corresponding consolidation decisions they make.

B.9. MATRIX ORGANIZATIONS

As explained before, the strict hierarchical organization method is based on a tree-like structure of control. Agents or agent teams report to a single manager, which provides the agents with goals, direction and feedback. Matrix organizations relax the one-agent, one-manager restriction, by permitting many managers or peers to influence the activities of an agent. This forms a mixed-initiative environment, where successful agents reason about the effects their local actions can have on multiple entities. This is in some sense a closer approximation to how humans exist. A person may receive guidance or pressures from their manager, co-workers, spouse, children, colleagues, etc. Even in a purely business setting one might have to report to an immediate supervisor, project managers, vendors, and peers at cooperating businesses. Interrelationships can come from many directions, each with its own objectives, relative importance and pertinent characteristics [Wagner and Lesser, 2000].

The term matrix organization comes from a grid based view of the participants. One can place managers (darker lower part) around a group of *"worker"* agents (clearer lower part), and use a directed edge to indicate

authority, as in *figure 44*. Alternately, agents are the rows and managers the columns (these sets may overlap), and a check is used to denote where an authority relationship exists. Like the hierarchy's tree, the matrix provides a graphical way to depict which managers can influence the activities of each agent.

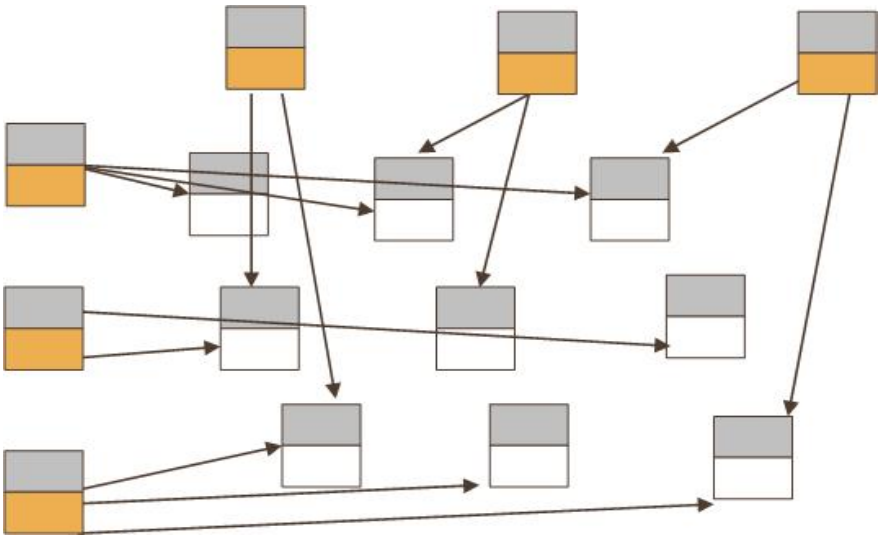


Figure 44. *A multi-agent matrix organization.*

B.9.1. CHARACTERISTICS

Matrix organizations provide the ability to explicitly specify how the behaviours of an agent or agent group may be influenced by multiple lines of authority [Decker *et al.*, 1995]. In this way, the agent's capabilities may be shared, and the agent's behaviours (hopefully) influenced so as to benefit all. This is particularly important if the agents themselves are viewed as functional, limited resources. For example, if a particular skill is needed by two separate tasks, the agent can be used to address both, provided it has sufficient computational power. In the case where the agent has multiple ways of performing a task, it can also choose the method which best satisfies its

peers.

This sharing comes as a price, however, because the shared agent becomes a potential point of contention. If its managers disagree, the agent's actions may become dysfunctional as it is pulled in too many directions at once [Schwaninger *et al.*, 2000, Romelaer, 2002]. To operate effectively, the agent must have a commitment ranking mechanism and sufficient autonomy to resolve local conflicts, or the ability to promote conflicts to a higher level where they may be resolved [Mailler *et al.*, 2003]. Wagner's motivational quantities framework [Wagner and Lesser, 2000] is one approach that addresses this problem. In that work, task valuation is performed by combining both the local intrinsic worth of the task with the perceived or specified worth that task will have on other entities. This valuation is quantified through the expected production and consumption of different motivational quantities (MQs), which act as a virtual resource or medium of exchange. The preference for particular MQs is specified with a set of utility curves that together determine the agent's overall usefulness. By coupling the production of different types of MQs with the tasks associated with different managers, the framework is able to capture the quantitative motivation behind a particular course of action. This explicitly represents the type and states of the relationships the agent has with those managers, which can enable it to correctly balance its behaviour in a matrix organization.

B.9.2. FORMATION

Decker [Decker *et al.*, 1995] describes the MACRON organizational architecture, in which agents form a matrix organization. The domain for their system is cooperative information gathering, where multiple agents search for relevant data in response to a user's query. Individual agents are separated into predefined functional groups that contain agents able to access a particular type of information. These groups are under the control of a functional

manager, who assigns agents to query tasks as they arrive. User query agents generate those query tasks, and therefore use the functional managers to dynamically select agents to satisfy their own goals. Individual gathering agents report to two agents: a static functional manager, and a query manager which changes depending on the user's actions. This has the effect of assigning the minimal needed set of agents to the query, increasing efficiency when compared to a system employing a set of static teams where particular team members might go unused, depending on the query characteristics. At the same time, this approach uses fewer resources than one lacking functional groups, which would have to search through all available agents for each query.

In [Horling, 2003], Horling describes a distributed sensor network application where a matrix organization is used to address a resource allocation problem. In this case, the sensors themselves were limited resources, since their heterogeneous locations and orientations made each one unique. The tracking process for each target was controlled by a different track manager, which was responsible for discovering and coordinating with the sensors needed to track its target. When multiple targets came in close proximity to the same sensor, a matrix organization is dynamically formed as the relevant managers interact with that sensor. At the same time, that sensor may have previously been given tasks by a regional manager responsible for detecting new targets.

The result is an individual which may be under contention by three or more managers, and which must then decide how best to meet those demands. This was done using a combination of a predefined ranking scheme (tracking has higher priority than scanning for new targets), local autonomy (round robin scheduling) and conflict elevation (track managers negotiate directly once aware of the conflict).

B.10. COMPOUND ORGANIZATIONS

Not all organizational structures fit neatly into a particular category, and some architectures may include characteristics of several different styles. A system may have one organization for control, another for data flow, a third for discovery, and so on. For example, Durfee's PGP [Durfee and Lesser, 1991] incorporates one organization for interpretation, and another separate structuring of the same agents to manage coordination problems.

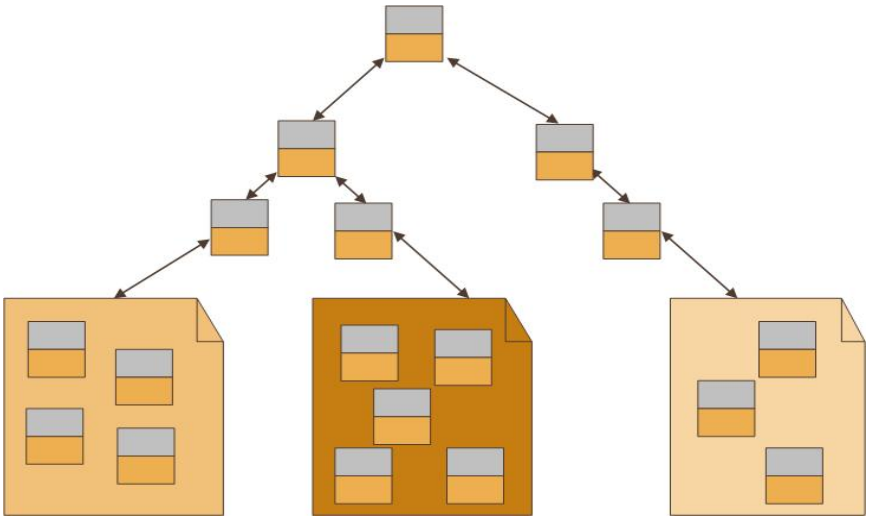


Figure 45. *A multi-agent compound organization.*

Compound organizations can be overlapped, operating as virtual peers at the same conceptual level, or be nested, so that some subset of agents in a group are organized in a potentially different way within the larger context. A sample such organization is shown in *figure 45*, which combines a hierarchy with a set of coalitions. As with singular organizations, they may be created or adapted over time, or they may be instantiated as part of a transient form while a population shifts between organizational styles. Ideally, these compound architectures can use the most effective structure for the particular goal at hand, without limiting options that might be used elsewhere in the system. The

trade-off in this situation is usually one of complexity. Because an individual agent might take on different roles in response to different organizational demands, the agent itself must have sufficient sophistication to act efficiently and asynchronously in all those roles.

Some of the organizational paradigms which have been discussed so far are more amenable to coexistence than others. In much of the teamwork research, for example, a loose hierarchy of control was created among the agents after the team had formed [Tambe, 1997, Tidhar *et al.*, 1996]. Hierarchical structures for interpreting and consolidating raw data are also a popular mechanism for handling scale that can augment a pre-existing or lower-level structure [Yadgar *et al.*, 2003].

Societies frequently have an internal organizational structure within the larger context defined by the social laws and norms [Dellarocas and Klein, 2000a, Dignum, 2004]. In other cases, researchers have exploited the characteristics of one type of organization to create another. Congregations, for example, have been used to facilitate the dynamic formation of markets [Brooks and Durfee, 2002], while both markets [Lerman and Shehory, 2000] and hierarchies [Abdallah and Lesser, 2004] have been used to efficiently create coalitions. Societies can also be viewed as a common "pool" of agents, from which a range of other organizations can be constituted. In this type of compound organization, the society may exist in support of other, more dynamic structures created to address particular tasks [Sichman and Demazeau, 2001].

B.10.1. CHARACTERISTICS

The positive and negative characteristics of a compound organization are derived primarily from its constituent parts. However, the interplay between organizations can lead to unexpected consequences. For example, if the distinguished intermediary in a federated system plays a key role in a

separate overlay organization, it may be unable to fulfil both roles adequately. Similar to a matrix organization, agents may be faced with conditions where it is not clear which of two competing objectives it should satisfy [Romelaer, 2002].

Conversely, its knowledge of the requirements of both organizations may enable it to make more globally effective decisions. The possible interactions and formation strategies among arbitrary coexisting organizations are difficult to characterize in a general manner; so some examples of systems employing this technique will be shown next.

B.10.2. EXAMPLE COMPOUND ORGANIZATIONS

The distributed sensor network solution described by Horling [Horling, 2003] uses several different overlapping organizational techniques. Agents are first partitioned into federations, called sectors, where membership is based on their geographic proximity. A distinguished member of each group is given the role of sector manager, who provides a form of recruiting service to other agents in the environment. This recruiting service supports the activities of track managers, who must discover and use the appropriate sensors as part of their tracking task. In forming the federations, the search time is reduced because only a subset of the population (the sector managers) needs to be interacted with, and communication requirements are reduced because only the necessary subset of sensors will be returned. Both the sector and track managers provide tasks to individual sensors, forming a matrix organization in the process. This arrangement facilitates resource sharing by allowing the sensors to guide their local activities based on the needs of potentially several interested parties, but can also lead to conflicts caused by over-demand. Because the sensor is a finite resource, a cloning technique cannot be used to address the conflict. Instead, a loose peer-to-peer relationship between track managers allows them to negotiate directly,

alleviating the conflict through demand relaxation or by using alternate sensors. This resource allocation scheme employs a second, weaker form of federation through its use of mediators [Mailler and Lesser, 2004].

The conflicts, which may be potentially multi-linked and far-reaching, are partially centralized by a mediator agent which acts on the part of the relevant agents to find a suitable solution. In [Horling *et al.*, 2004] the quantitative effects of these interactions are demonstrated through a set of experiments that vary the shape of the organizational structure.

Yadgar [Yadgar *et al.*, 2003] describes a different approach in a distributed sensor environment. Groups of geographically-related sensors are first formed into sampler groups, which are essentially federations with a single agent called the sampler group leader acting as the intermediary. These groups then form the lowest level of a data aggregation hierarchy that exists above them. This arrangement is similar to the example organization shown in *figure 45*. The sampler group leader collects raw data from the members of its group, and passes the data to its parent agent in the hierarchy, known as a zone leader. It is this zone leader's responsibility to interpret the sensor data to the best of its ability, by building motion equations and combining data perceived to be from the same target. This more abstract view is then passed to the next level of the hierarchy, where the process repeats. This will eventually terminate at the apex agent which should be able to reconstruct a global view from the abstract pieces it receives. The hierarchy itself is strict, and communication is only permitted between connected agents, which reduce the level of sophisticated needed by the agents.

The experimental results showed that this solution could scale to thousands of sensors and targets. The trade-off they discovered was that shorter hierarchies produced more accurate results, because the fragmentation of the area was minimized, which in turn reduced the number of fusion processes data must survive before it is incorporated. Conversely, taller

hierarchies dramatically reduced the computational load placed on any one agent, because the area each agent was responsible for became relatively small. By weighing these characteristics against the domain requirements one can select an appropriate structure to use.

B.11. OTHER ORGANIZATIONAL TYPES

There are a number of other topics related to organizational design that, although they are not so widely used, they are sufficiently important to warrant mention. These are outlined below:

Global Organizational Representation. Implicit in the concept of an intentional organizational design is an explicit representation of its structure. This is of use to designers, as a means of specification and exploration, and to the agents themselves, as a template and diagnostic tool. A number of general modelling representations have been proposed, notably by Fox [Fox *et al.*, 1998], Tambe [Tambe *et al.*, 1999], Hübner [Hubner *et al.*, 2002], Pattison [Pattison *et al.*, 1987], Dignum [Dignum, 2004], Sims [Sims *et al.*, 2004], Horling [Horling and Lesser, 2005] and Vázquez-Salceda [Vázquez-Salceda *et al.*, 2005].

Local Organizational Representation. The organization's global view is not always the most appropriate vehicle to guide agents' behaviours. It can be too coarse in granularity, too qualitative or simply too large to be of practical use. Agents require a well-defined, quantitative mechanism that can be used to select appropriate local actions while respecting global organizational specifications. This process was originally described as local elaboration by March and Simon [March *et al.*, 1958], where the activities performed by an agent are first constrained by its position in the organization, and then selected using local information and capabilities. The social consciousness model suggested by Glass and Grosz [Glass and Grosz, 2003], Decker's TÆMS language [Decker and Lesser, 1993], Shoham's social laws

[Shoham and Tennenholtz, 1995], and Wagner's MQ framework [Wagner and Lesser, 2000] provide ways to accomplish this.

Organizational Performance. Other researchers have taken a different approach by creating formal analytic or statistical models that focus on the activities or behaviours of the organization, rather than representing the organization as a whole [Malone and Smith, 1988, Decker and Lesser, 1992, Montgomery and Durfee, 1993, So and Durfee, 1996, Lerman and Galstyan, 2001, Shen *et al.*, 2004, Gnanasambandam *et al.*, 2004, Horling and Lesser, 2005, Schmitt and Roedig, 2005]. These typically more quantitative representations can provide insights into organizational performance that are largely absent from purely descriptive or logical representations. A different approach is to use experimental or simulation studies, which can offer a more general-purpose approach to analyze organizational performance that may not be amenable to modelling [Lesser and Corkill, 1983, Lin and Carley, 1995, Sierra *et al.*, 2004]. The drawback to using empirical analysis is the time required to run such tests, which is usually much greater than that needed for analytic techniques. Conversely, analytic models may require simplifying assumptions to be tractable, or otherwise fail to take into account the complexity real-world behaviours. Parunak [Parunak *et al.*, 1998] provides further discussion on the tradeoffs between these approaches. However they are obtained, such predictions can play a critical role in the search and evaluation process, by allowing the designer to directly compare alternative organizational strategies before implementing a design. This can provide the foundation for a more proscriptive organizational tool.

Generative Paradigms. Different ways in which organizations may be formed have been described before. However, it has not been presented a unified discussion of specific generative paradigms – a classification of the techniques that may be used to produce organizations. These may be broadly separated into at least three classes: scripted, controlled and emergent. The

first includes organizations that are produced from statically predefined instructions, possibly from an external third party or during start-up. The second includes those that are explicitly applied to a population by an individual or group of individuals in response to perceived conditions. The third captures techniques which have no central or global direction, but are instead self-directed or grown organically through the individual actions of agents. In practice, it may be difficult to clearly classify particular techniques. For example, congregations emerge from individual agent decisions using the technique described by Brooks [Brooks and Durfee, 2002]. However, the fact that it uses heuristics intended to simulate a controlled decision, along with agents which provide labels to guide the formation, gives the appearance of a controlled process.

Organizational Adaptation. Although adaptation has been previously briefly touched, an organization's ability to adapt is a general concept that is critical in any dynamic environment. The organization must have the ability to detect and react to changes in a timely manner in realistic, open domains [Carley, 1997, Horling *et al.*, 2001]. Any organizational change which occurs at runtime will have associated costs. These costs may be observed in direct consumption of resources, such as bandwidth or processing power, or indirectly because of inefficiencies or opportunities missed while in an intermediate state. The ability to adapt an organization depends on first recognizing potential problems, evaluating the costs and benefits of candidate solutions, and then implementing the selected changes. Related to adaptation is the notion of social pathologies, which occur when an organization adapts inappropriately [Turner, 1993, Jensen and Lesser, 2002].

Coordination and Negotiation. Many of the organizational styles covered assume some that some sort of interaction or coordination will take place between agents. This is seen in the authority relationships of hierarchies, the joint intentions of teams, data routing protocols in federations, and

negotiations of society members. The characteristics provided by these interactions are critical to the effective qualities of these paradigms. For example, aggregating nodes and managers in hierarchies and intermediaries in federations frequently take on responsibilities related to coordination, by assigning tasks or routing information in such a way that interrelationships among their subordinates can be avoided [Galbraith, 1974]. Argumentative negotiation has been shown to be effective in resolving conflicts in team settings [Jung *et al.*, 2001]. The techniques that are used can heavily influence the interactions and behaviours exhibited by the group, ultimately affecting the performance of the organizational structure. Work by Prasad [Nagendra Prasad and Lesser, 1999], Lesser [Lesser *et al.*, 2004] and Toledo [Excelente-Toledo and Jennings, 2004] have also explored the dynamic selection of coordination strategies, which in this context can be considered a form of organizational adaptation.

Autonomy. The manner in which an agent behaves, and in particular how its motivations are determined, is intimately related to its position within the organization. Agents may be externally directed, self-directed or some combination of the two [Lesser and Corkill, 1981]. For example, agents in hierarchies, federations and matrix organizations all generally have manager-supervisor relationships, implying that local actions are partially or completely decided by an external entity. Conversely, agents operating in markets are typically more autonomous, independently deciding how and when to bid. Like other characteristics, the level of autonomy can affect the performance of the system as a whole. Authoritarian structures can exploit centralization to make good decisions, while an organization of more autonomous entities offers better balance and parallelism. Because the needs and constraints exhibited by participants change over time, it can also be beneficial to dynamically adapt agents' levels of autonomy in response to changing events [Scerri *et al.*, 2002, Zhang *et al.*, 2002].

Human Organizational Analogues. For much of the time that multi-agent organizations have been researched, attempts have been made to draw upon the large body of work that has been done on human organizations. The fields of sociology, anthropology, biology, economics, business management and formal organization theory (among others) contain a wealth of analytic and case study information describing how human organizations are structured and perform [Fox, 1981, Gasser, 2001]. Although on the surface much of this work is intimately tied to the human experience, attempts to extract concepts and abstractions have met with some success.

Diversity. Although role assignment clearly plays a critical role in an organizational specification, the notion of agent diversity is rarely treated as or reasoned about as a first-class characteristic. As with stock portfolios, animal populations and security techniques, diversity can play an important role in agent systems susceptible to failure. Enforcing agent diversity through heterogeneous roles, agent types or division of labour, can impart semantic and capability fault-tolerance on the system as a whole [Corkill and Lesser, 1983, Reed and Lesser, 1980, Corkill and Lander, 1998, Lybäck, 1999]. Diversity can be embedded in the organizational design to encourage such characteristics.

”Each problem that I solved became a rule which served afterwards to solve other problems.” *René Descartes*

Appendix C. Case-Based Reasoning

In this appendix the Case-Based Reasoning methodology is introduced. CBR is the core methodology of the OBaMADE architecture, being responsible of the structure of the stored information and of the quality of the results. The CBR methodology is used to generate the solutions by reusing past solutions given to past problems. The four main phases of the CBR cycle are explained here, paying special attention to the CBR systems developed based on this methodology.

Case-Based Reasoning is a methodology that has its origin in knowledge based systems. CBR systems learn from previous situations [Aamodt, 1991]. The main element of a CBR system is the case base; a structure that stores problems, elements (*cases*), and its solutions. So, a case base can be visualized as a database where a collection of problems is stored keeping a relationship with the solutions to every problem stored, which give the system the ability to generalize in order to solve new problems.

The learning capabilities of the CBR systems are due to its own structure, composed of four main phases [Aamodt and Plaza, 1994]: *retrieval*, *reuse*, *revision* and *retention*. These four main phases are shown in *figure 46.* The first phase is called *retrieve*, and consists in finding the most similar cases to the proposed problem from the case base. Once a series of cases are extracted from the case base, they must be *reused* by the system. In this second phase, an adaptation of the selected cases is done to fit the current problem is done to fit the current problem. After giving a solution to the problem, that solution is *revised* to check if the proposed alternative is a solution to the problem. If the proposal is confirmed as a solution, then it is *retained* by the system and could eventually serve as a solution to future problems.

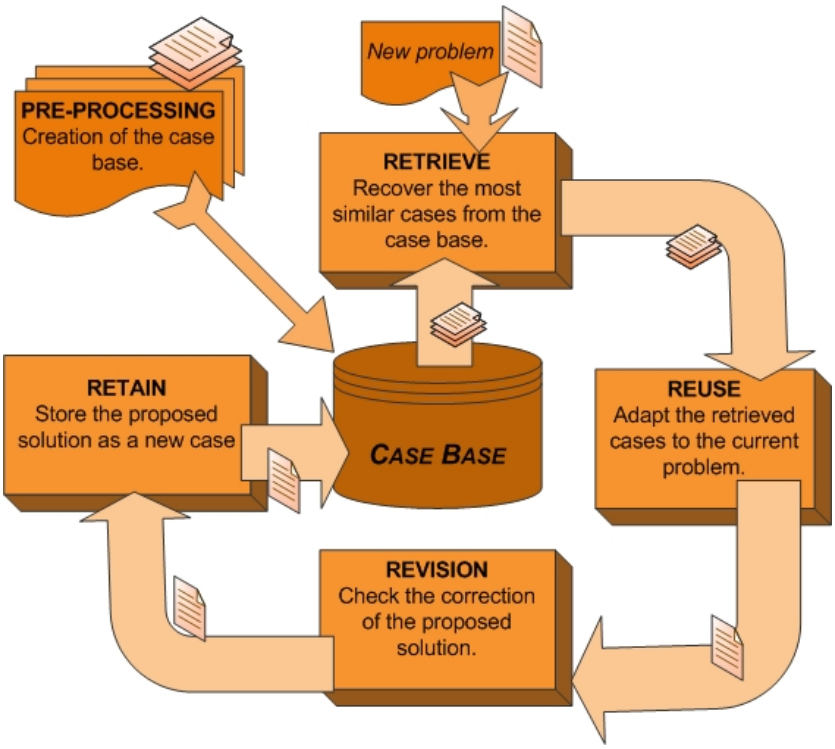


Figure 46. *Case-Based Reasoning basic structure.*

Case-Based reasoning is a *methodology* [Watson, 1999], and so it has been applied to solve different kind of problems. It is a model that can be easily applied to solve soft computing problems [Shiu and Pal, 2004], since the methodology used by CBR is quite easy to assimilate by soft computing approaches. Another interesting application is related with stock market prediction [Chun and Park, 2005], where using different daily values, a CBR system can create a model that may help in stock market investments. Construction is another of the fields of application of CBR, first for the construction of functional databases [Yu and Liu, 2006] to improve the benefits in the usually chaotic organization of the construction projects and also [Chow *et al.*, 2006] to help to choose between different methods and materials, using expert system oriented applications.

Other applications of the CBR methodology cover from health applications [Corchado *et al.*, 2008] to eLearning. CBR has evolved, being transformed so that it can be used to solve new problems, becoming a methodology to plan, or distributed version. Oceanographic problems [Fdez-Riverola and Corchado, 2004], has also been solved with these techniques, helping to predict the value of variable parameters.

But, in most cases, CBR has not been used alone, but combined with various artificial intelligence techniques. Growing Cell Structures has been used with CBR to automatically create the intern structure of the case base from existing data and it has been combined with multi-agent applications [Carrascosa *et al.*, 2007] to improve its results. ART-Kohonen neural networks [Yang *et al.*, 2004],, artificial neural networks and fuzzy logic [Fdez-RiverolaIglesias *et al.*, 2007a] has also been used to complement the capabilities of the CBR methodology. Actual trends in CBR explore the possibility of giving explanations from the very CBR systems [Sørmo *et al.*, 2005]. These techniques allow the CBR systems to give the users a better solution, adding extra information to the solution proposed by the system.

C.1. CASE-BASED REASONING AS A PROBLEM SOLVING APPROACH

Reasoning can be defined as a process that draws conclusions by sequencing generalized rules or situations. The principal knowledge source of CBR is not generalized rules but a memory of stored cases. In CBR, new solutions are generated not by chaining but by retrieving the most relevant cases from case library and adapting them to fit new situations [Leake, 1996].

CBR tasks are often divided into two classes as interpretive CBR and problem-solving CBR. Interpretative CBR uses prior cases as reference points for classifying or characterizing new situations; and problem-solving CBR uses prior cases to suggest solutions that might apply to new circumstances [Kolodner, 1993].

The interpretive CBR involves four steps being performing situation assessment [Kolodner, 1993] to determine which features of the current situation are really relevant; retrieving a relevant prior case or prior cases based on the results of situation assessment; compares those cases to the new situation and finally saying the current situation and the interpretation as a new case for future reasoning [Leake, 1996].

Legal problems and diagnosis concepts are the fields for which interpretive CBR processes are applied. On the other hand, in problem-solving CBR, the goal is to produce a solution to a new case based on the adaptation of solutions to past cases. Case-based design, planning, and explanation systems are the examples for this class since they require retrieving and adapting solutions of similar prior problems [Leake, 1996]. Like interpretive CBR, problem-solving CBR involves situation assessment, case retrieval, and similarity assessment steps to find solutions for new

problems. Since many problems have components of both types of CBR, most effective case-based reasoning systems use a combination of both methods [de Mántaras and Plaza, 1997].

In short, CBR solves problems through a process that involves some basic steps as retrieving relevant cases from the case memory, selecting a set of best cases, deriving a solution, evaluating the solution and storing the newly solved case in the case memory [de Mántaras and Plaza, 1997].

The goal of CBR is to use the computer to augment the analogical reasoning and memory of the domain expert by providing the expert with representative cases similar to the problem at hand [Kolodner, 1991]. This statement points out the necessity of computers to apply CBR principles. In order to meet this requirement, several commercial companies offer shells for building CBR systems. CBR shells provide mechanisms to support case retrieval and allow users to interactively provide additional information as needed during retrieval besides; they provide sophisticated interfaces to facilitate creating and editing the case base [Leake, 1996].

C.2. CASE DEFINITION AND CASE BASE CREATION

The first phase in the design of a CBR application must consist in a transformation of the information available into a structure, into cases. This transformation is a crucial step in the creation of a good solution. Not all types of information can be easily traduced into cases and so, the possible variations can dramatically modify the correction of the solutions proposed by the systems.

A case can be defined as a conceptualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner [Kolodner, 1993]. It is a set of features, attributes and

relations of a given situation and its associated outcomes. Case acquisition is an important aspect in designing efficient CBR systems. Cases in the case memory are designed to capture the knowledge and experience of domain experts [Gupta, 1994].

Cases are collected in a database which is composed of cases with each case including; a set of problems, characteristics that distinguish this set from others that warrant a different response, possible actions that were particularly helpful or harmful in such situations, indicators that suggest what type of response to expect and connections to other cases that reflect next steps or alternate steps depending on the responses observed [Kolodner, 1993]. Since the case base reflects the conceptual view of the cases and it supports efficient search and retrieval methods, it should be organized in a manageable structure, which determines the scope of intelligence of the system and its breadth and depth of expertise [Gupta, 1994].

One of the main concerns of CBR is to ensure that the right cases can be recalled at the right times. This is known as the indexing problem in CBR, which has two aspects. One is the vocabulary problem that requires assigning suitable labels or descriptors to the case so that it can be easily referenced in the case library during retrieval [Chua *et al.*, 2001]. Indices should address the purposes the case will be used for; they should be abstract enough to allow for broadening the future use of the case base and concrete enough to be recognized in future. However, despite the success of many automated methods, Kolodner [Kolodner, 1993] believes that people tend to do better at choosing indices than algorithms, and therefore for practical applications indices should be chosen by hand.

A CBR system uses a set of indices to search for and retrieve cases similar to the current problem. There are three main approaches in indexing cases namely nearest neighbour, inductive reasoning and knowledge guided indexing [Gupta, 1994]. Frequently, systems use a combination of all three

methods. In the nearest neighbour approach, the system selects the case whose attributes most closely match those of the current problem. Among current machine learning methodologies, inductive learning is the most widely used.

An example of inductive learning systems is ID3 [Li, 1996], which the majority of the case-based systems implement. The objective of induction algorithm is to generalize decision rules from past examples. These methods use an intelligent approach to retrieve cases based on the most meaningful and discriminating features of each case.

On the other hand, in knowledge-based indexing, domain knowledge about each case is used to determine the features in past cases that are most relevant to the current problem. This method is generally used to enhance and supplement the other two indexing approaches due to the difficulty to implement this method since explanatory knowledge cannot be successfully and profoundly captured using if-then rules [Gupta, 1994].

The easiest way to create a case is just a series of numerical values [Tsai and Chiu, 2007] that correspond to those variables that are going to be considered as important in order to solve the problem. When the characteristics of a system can be expressed as numbers [Pérez *et al.*, 2005] it is quite easy to generate a case structure that can be used by mathematical techniques.

In other cases, properties of the variables that must conform the case are selected [Song *et al.*, 2007] to easily transform information into cases, measuring and transforming the properties in order to clearly obtain the information that is useful for the developed application.

In textual case bases, it is sometimes necessary to extract knowledge from the data before creating the case base [Mustafaraj, 2007]. Once the knowledge is obtained, it can be structured into the case base. Every new element is part of one or more of the pieces of knowledge previously

identified, and then, the case is formed by the separated pieces that has inside it.

E-mails are also textual elements [Fdez-RiverolaIglesias *et al.*, 2007b], and the transformation from information to cases is not always obvious. If the most relevant terms are selected, it is necessary to determine which terms are more relevant than the others, and to justify it. A set of mails is used and then a comparison between the frequency of appearance of a term in a message and the frequency of the same term in the whole set of mails is established as a measuring value.

In medical applications, the case must include values referred to the patient, but also associated with the clinical evolution of the patient [Montani, 2007]. It is also interesting to include a reputation value that is increased every time a case is recovered from the case base and used, every time the expert considers that the case is useful.

When the information to be transformed into cases contains a great amount of words, it is necessary to parse the original data [Patterson *et al.*, 2005] in order to obtain the list of terms used to create the cases.

In some occasions, the information can be considered as hard to model, but after an analysis, it can be transformed into numerical variables [Ros *et al.*, 2006] with what is quite easier to generate cases.

There is a clear difference between cases related with textual information and those where the information can be numerical. In textual systems a filtering process must be produced in order to eliminate useless information and to traduce the data available into a series of concepts that can categorize every item in the case base. On the other hand, numerical information has a clear representation into cases, but, sometimes, it is not evident and the variables must be evaluated, confronted or even transformed.

C.3. RECOVERING DATA FROM THE CASE BASE

Once the information is stored is the case base, it will be used to solve future problems. The case base store all the cases previously used by the system. When a new problem appears, a selection of cases are recovered from the case based and will be used to solve that new problem.

The cases retrieved from the case base are in most cases, those more similar to the proposed problem. Similarity is those systems the key concept to take into account when trying to improve the retrieval phase, but it is not the only valid concept in order to improve the retrieval.

The indexing mechanism determines the cases that should be selected while the case retrieval process ensures that the most relevant case is selected for further analysis. Given a description of a problem, a retrieval algorithm retrieves the most similar cases to the current problem or situation by using the indices in the case library. The retrieval of relevant cases depends on a good indexing of the cases that select an appropriate set of indices. The system retrieves the matched cases according to a predefined similarity function, which evaluates the degree of similarity of each case in the case base [Yau and Yang, 1998a].

CBR systems should include a strong memory-based retrieval system; cases should be retrieved intelligently and systematically by finding the closest match between attributes of past cases and those of the current problem [Gupta, 1994]. When the case memory is large, a hierarchical organization of the memory is necessary because a simple linear list is very inefficient for retrieval. The basic idea is to organize specific cases that share similar properties under a more general structure called a “generalized episode” [de Mántaras and Plaza, 1997]. A general episode contains norms,

cases and indices where norms are features common to all cases, indexed under a general episode and indices are features, which discriminate between the cases of a general episode [de Mántaras and Plaza, 1997].

One of the most famous similarity measures is the k-NN (k nearest neighbours) and also modern variations like Significant Nearest Neighbour [Tsai and Chiu, 2007] where the value of k is calculated taking into account the dissimilarity between the new case and the past ones stored in the case base.

In some cases, when the amount of variables is quite big, it is necessary to select which ones will be used to select the similar cases from the case base [Montani, 2007]. A two steps procedure occurs so first the interesting variables must be chosen, and then, the search in the case base of the most similar cases according to those variables.

To determine the similarity between different elements, a great variety of metrics has been used. Sometimes it is recommended to establish the similarity between two elements by comparing them with the rest of the cases [Im and Park, 2007]. Then the compared elements will be considered as similar if their similarity with the rest of the cases is similar in all cases.

If different features are considered when defining the case base, they must all be considered when obtaining similar cases from the case base. In this kind of situations different metrics can be done to calculate the similarity of the different features [Ros *et al.*, 2006], and then create a combined similarity metric that integrates all the metrics used.

Recover the most similar cases to one given can be an easy task if the whole case base is indexed [Galushka and Patterson, 2006], then it is only a question of searching the closest cases. But to get to that point, a previous effort of analysis and categorization of the information must be done.

In some circumstances, a previous search of context is done [Spasic *et al.*, 2005], to obtain a variety of cases that are used to perform a second and

more specific search.

When facing textual problems it is interesting to offer different alternatives so that the user can personalize the retrieval depending on the interest of the query [Patterson *et al.*, 2005]. This way, the recovered cases can be adapted to a specific situation defined by the user when determining the terms of the retrieval.

When the different variables stored in the case base represent a dissimilar importance for the final solution, it has to be expressed in the way the cases are retrieved from the case base[Nugent and Cunningham, 2005]. The importance of the variables may also vary from one query to another, and so the retrieval system must be adapted to correctly get back the right collection of cases from the case base.

If the problem introduce in the system implies considering different scenarios, multiple retrievals can be done [Aha *et al.*, 2005]. In this kind of situations the original problem introduced in the system defines the start point of the search, and from that point and looking for in different directions, different sets of cases are recovered from the case base, in order to generate a complete perspective of the problem.

C.4. ADAPTATION OF THE RETRIEVED CASES

The reuse phase is the solution generator. From the collection of cases retrieved from the cases base, a new solution must be generated in order to solve the proposed problem. Sometimes, there is no need to modify the recovered cases to solve the problem, especially if talking about classification problems, where only a belonging solution must be offered.

The most complex the problem is, the most necessary an adaptation is. When the difference between the introduced problem and the stored cases is

big enough, then the adjustment of the recovered cases is essential in order to obtain a correct solution, really adapted to the proposed problem.

Once a matching case is retrieved, a CBR system should adapt the solution stored in the retrieved case to the needs of the current case. In general, there are two kinds of adaptation in CBR as structural adaptation in which adaptation rules are applied directly to the solution stored in cases and derivational adaptation that reuses the algorithms, methods or rules that generated the original solution to produce a new solution to the current problem [Kolodner, 1993].

Most research on case adaptation has assumed that adaptation should be done in a completely autonomous way through the rules. There are alternatives of decreasing the need for adaptation rules suggested by Leake [Leake, 1996], some of which are using flexible adaptation rules, using adaptation cases, combining rules and cases for adaptation learning and reusing subcases. Adaptation rules as proposed by Ng [Ng, 2001] are developed to guide the adaptation process.

The next step after a case is adapted in accordance with the requirements is the incorporation of that case into the case base so that it can be used in the future. This feature of CBR provides the algorithm to become stronger since the following problems will be solved more accurately with a larger database. If the proposed solution is successful then the system incorporates the solution and the representation of the current case into the case memory. Sometimes, the system may not propose a solution to the problem. In such cases, if the solution fails, then the system provides an explanation as to why it failed and documents it in the system library [Gupta, 1994].

The reuse phase implies adapting the retrieved cases to solve the new problem. In some cases multiple adaptations can be done [Huang *et al.*, 2007], depending on the amount of information given to the system. The

biggest amount of information given, the most direct transformation will be done.

When treating textual information, like e-mails, voting algorithms [Fdez-RiverolaIglesias *et al.*, 2007b] can be used to adapt the recovered cases, taking into account the information proposed by the treated problem.

On the other hand, numeric situations, like those used in microarray problems, can be reused thru neural networks like Growing Cells Structures [Diaz *et al.*, 2006], where the aim is to cluster the retrieved information.

Another way to use neural networks to adapt the retrieved information is to change the weight of the connection between the neurons depending on the retrieved cases [Zhang *et al.*, 2004]. Changing the weights allows the system to adapt the solution to the problem, as the retrieved cases will depend directly on the proposed problem.

When the certitude about the correction of a solution is not high enough, multiple cases may be taken into account in order to build the new solution. Then a fusion of cases [Song *et al.*, 2007] is done, considering the different benefits given by every point of view, by every case retrieved.

If the problem to be solved may belong to more than one field of knowledge, and there may be more than one case base, a good solution can be to adapt the retrieved cases, from the different case bases, according to the characteristics of the problem [Policastro *et al.*, 2006]. In this case, neural networks were used to recover the data from the different case bases, and machine learning algorithms combined the retrieved cases in order to adapt those cases to the proposed problem.

When using genetic algorithms, the reuse may help to reduce convergence time if considering previously working solutions [Pérez *et al.*, 2005]. This approach may be applied to different fields where evolutionary algorithms are useful but slow.

C.5. REVIEW OF THE PROPOSED SOLUTION

When a solution is generated by a system, it is necessary to validate the correction of that solution. One easy way to validate that correction is to compare the proposed solution with those stored in the case base [Yu and Liu, 2006]. Then a threshold value is established in order to determine if the new solution is correct enough to be considered as a good solution and so to be stored in the case base for future uses.

If the case base structure is integrated into a neural network, then the revision phase consists changing the organization of the case base, depending on the correction of the proposed result and other neural variables such as neuron age, activation value and last use [Wu and Yu, 2005].

The best way to test the correction of a solution is to actually perform the solution and check how good has been the evolution after applying it. This is only possible in certain environments, such as strategy games [Aha *et al.*, 2005], where what is analyzed is the tool and its algorithms.

In crucial fields, such as medical applications, it is normal to trust an expert in order to finally accept a solution [Chang, 2005]. Then, after being accepted by the corresponding expert, next time it will be considered as a better solution, being chosen from the case base with a higher probability.

Changing the values proposed by the system to others similar but not equal is a technique also used to revise the correction of a solution [Li *et al.*, 2007]. If the solution generated by the similar values is not better than the proposed one, then the chosen one is a good solution for the problem.

In not critical applications, like strategy games, the correction of a solution can be added to the stored solutions, increasing its value every time a solution is chosen [Sharma *et al.*, 2007]. Genetic algorithms are also used to revise the correction of the solutions [Pavón *et al.*, 2008]. After running those algorithms, the solutions can be accepted, and added to the case base.

Finally, fuzzy algorithms are also used to automatically revise CBR solutions [Fdez-RiverolaDíaz *et al.*, 2007]. Using those algorithms the memory used to store the cases can also be reducing, improving the result of the system.

C.6. RETAIN OF THE SOLUTION AND CASE BASE MAINTENANCE

The retention phase is a very important element in the case base maintenance [Wilson, 2001]. It is important to readapt the way the information is stored in order to increase the possibilities of finding good solutions in the future. New data may affect previous relations established between the stored elements. So it is important to arrange solid criteria to decide whether to change the case base or not and if so, how to do it correctly in order to represent in the case base the whole variability of the available data.

In most cases there is a big amount of information stored in the case base and it is not necessary to store every valid case, thus the information could be too redundant. In those situations a conditional retention is performed [Sharma *et al.*, 2007], keeping the new solution only if it is different enough to the closest existing case.

If during the solving process a big amount of new information is generated, it may be eventually introduce in the case base. The relevance of the new information could be such to also affect the adaptation phase [Li *et al.*, 2007]. In those circumstances the retention process is not very strict because of the variety of origins that new data can have.

There are special applications where the source of new cases is not only the solution proposed but also information exchanged between different elements of the system [Ontañón and Plaza, 2003].. Then, the retention must

consider more variables, not only variability, but also the confidence or not of the transmitted data depending on the specific context.

Even when the proposed solution is considered as an eventually good solution to be stored in the case base, the growth of the case base can be counterproductive. In some case, where the amount of stored information is huge and when there must be an economy of resources in order to manage a reasonable case base, case base editing is necessary [Delany, 2006]. In those situations the number of cases stored in the case base is tried to keep as low as possible, always maintaining the inherent capabilities of the information.

When the case base grows to thousands of elements, it may be difficult to maintain it. Then dividing the case base in different parts with certain inner similarity [Li *et al.*, 2006] can help to structure the store information and also to make future retrievals.

Another strategy used to control the growth of the case base is to group cases into prototypes that [Montani and Anglano, 2008] include the common characteristics of a series of cases with no plenty of variability. Using those prototypes, the final size of the case base is reduced without losing a significant amount of information.

C.7. CASE-BASED REASONING COMPARED WITH OTHER TECHNIQUES

Reasoning in CBR is based on experience or remembering. CBR approach focuses on how to exploit human experience, instead of rules, in problem solving and thus improving the performance of decision support systems [Chen and Burrell, 2001]. CBR does not require an explicit domain model; main task is gathering case histories since CBR systems can learn by acquiring new knowledge. Identifying significant features to describe a case is much easier than creating an explicit model. By utilizing database

techniques, CBR is enabled to manage large volumes of information that increases the reliability of the solutions it proposes. Case-based systems are preferable when the expert knowledge is hard to be modelled and large amounts of cases are available. In this respect, case-based systems that aid problem solving in construction are assumed to be attractive as they provide a model to store previous construction projects in entirety as cases and reuse them when similar new problems occur [Li, 1996].

There are several alternative approaches in the AI domain over which CBR has various advantages. These systems include artificial neural networks (ANNs), rule-based expert systems and model-based systems. Rule-based systems have well-defined structures and excellent explanation facilities; in this respect they are more advantageous compared to ANNs, which cannot easily generate explanations for their results. Indeed, combination of rule-base systems or model-based systems with CBR could give more satisfying results since the strengths of one system may compensate the weakness of another.

CBR allows decision makers to interact with and review the reasoning process and even perform heuristic adjustments on the derived result where necessary [Chua *et al.*, 2001]. CBR is applicable to solve problems and make decisions when the knowledge needed is so vague that formatting decision rules is infeasible but cases are available [Li, 1996]. CBR eliminates the bottlenecks of other systems and facilitates development of expert systems. It benefits from how humans reason and it is based on experience, which should not be necessarily transformed to rules or models; it addresses ill-defined problems by tolerating human interpretation, which provides acceptable explanations on the solutions derived. Following paragraphs give a detailed analysis of each technique and discuss their similarities with CBR and the discriminating features between those methods.

C.7.1. ARTIFICIAL NEURAL NETWORK

An ANN is a computer program that imitates human decision making at a low level in an attempt to replicate the capacity of human reasoning to surpass the structure of rigidly defined rules and formal logic [Li, 1996]. A more comprehensible definition is given by Caudill and Butler (1990) who define ANN as a type of information processing system whose architecture is inspired by the structure of biological systems [Arditi and Tokdemir, 1999].

The development of an ANN based system consists of designing and training the ANN. The design parameters in constructing an ANN model can be described at three different levels: node level (type of input accepted, transfer function and means of combination), network level (number of layers, number and type of nodes, size of hidden layers, number and type of output nodes and connectivity) and training level (learning algorithm and learning parameters) [Arditi and Tokdemir, 1999]. Unfortunately, there is no structured methodology for designing an ANN [Li, 1996]).

Training consists of presenting input and output data to the network [Arditi and Tokdemir, 1999]. For each example presented to the network, outputs are produced and these outputs are compared with those expected. The error is back propagated to the hidden units and the weights of the connections are modified using a modification rule [Li, 1996]. The process is performed many times until the error is reduced to a preset level.

Obviously, there are some similarities between two approaches. Both are based on the experiential knowledge and are designed by acquisition of inputs and outputs to the system. It should be noted that CBR is a more advanced approach, it allows human interference in

deciding indexing methods, but ANNs work like a black box [Yau and Yang, 1998b], as the algorithm cannot be understood completely by humans.

In addition, ANNs require to be completely trained; they perform at lower efficiency when there are many features and do not allow updating the system without retraining, so they can be regarded as difficult systems to develop. Another drawback of ANNs is that they are designed to deal with only numerical figures. On the other hand, CBR systems seem to be more flexible since they are good at handling missing data, incorporating new cases into the case base and coping with a vast amount of features due to the indexing abilities. ANN is useful in identifying underlying patterns to be used for forecasting where available data are noisy and complex [Li, 1996] so, construction cost estimation may be an application area.

C.7.2. RULE-BASED EXPERT SYSTEMS

Expert systems are computer programs that use heuristics and inference techniques to solve complex problems that ordinarily require expertise [Gupta, 1994]. A rule-based expert system consists of a knowledge base to store the expert's knowledge and facts as rules, an inference engine that facilitates a reasoning process to solve a specific problem, a context memory that contains the information about the problem to be solved and a user interface that inputs and outputs information [Li, 1996].

The essence of an expert system is a knowledge base represented primarily by transparent if-then rules, so it is limited by the process of acquiring knowledge. Moreover, in most cases, an expert system cannot learn and has an extremely limited tolerance of incomplete input information when the system's default values are inadequate to solve the

new problem [Yau and Yang, 1998b].

Expert systems and CBR have a common goal of enhancing the intelligence of machines and making them more human-like. One important distinction is that expert systems solve problems by deductive reasoning from first principles [Gupta, 1994] whereas CBR systems solve new problems through analogical reasoning using the knowledge gained from past experiences.

Instead of relying solely on general knowledge of a problem domain or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations [Aamodt and Plaza, 1994].

As a CBR system modifies its behaviour based on past learning experiences, it may be assumed to be a more dynamic approach than rule-based expert systems, which are based on strict if-then rules. This is supported by Kolodner [Kolodner, 1991] who believes that expert systems are unsuccessful in solving problems that require creativity and common sense but case representation sometimes overcomes such problems. CBR systems are preferred over expert systems if rules are inadequate to express the richness of the domain knowledge.

C.7.3. MODEL BASED SYSTEMS

In model-based systems the actual performance of a process or task is compared with predicted behaviour or expected performance [Li, 1996]. Model-based reasoning uses structural knowledge of the domain in problem solving; it provides causal explanations; lead to robust and flexible problem-solving and allow transfer of some knowledge between tasks since science strives for generally applicable theories.

Besides these strengths, some disadvantages may be regarded as lacking experiential knowledge of the domain; requiring an explicit domain model; being highly complex and being unable to handle exceptional situations [Luger, 2002]. Model-based systems are beneficial for diagnosing problems for which a complete and accurate mathematical model exists [Li, 1996]. In contrast, CBR does not require extensive analysis of domain knowledge and it enhances problem solving through the indexing strategies.

”Sólo el que ensaya lo absurdo es capaz de conquistar lo imposible” *Miguel de Unamuno*

Apéndice D. Resvmen de la investigación

Este último apéndice del presente documento representa el resumen, en castellano, de la investigación mostrada a lo largo de esta tesis doctoral. Se detallarán, de manera sucinta pero efectiva, los distintos pasos que ha ido siguiendo esta investigación, así como los elementos previos necesarios y los resultados generados. De esta forma, se cubrirá todo el ciclo de vida de la investigación, desde los primeros requisitos iniciales hasta la evaluación de los resultados.

Este apéndice final, va a mostrar de forma resumida, los distintos elementos que han conformado la investigación plasmada en la presente tesis doctoral. Se muestran aquí los pasos llevados a cabo para, finalmente, desarrollar la arquitectura que se presenta y demostrar su validez aplicándola a dos casos de estudio diferentes.

La arquitectura presentada en este documento tiene por nombre OBaMADE (Organization Based Multiagent Architecture for Distributed Environments: *Arquitectura Multiagente Basada en Organizaciones para Entornos Distribuidos*). Se trata, como su propio nombre indica, de una organización *multiagente*. Dentro de los distintos tipos posibles de sistemas multiagente, se ha elegido una estructura basada en *organizaciones*, dándole especial énfasis a la capacidad de los agentes para trabajar conjuntamente, teniendo un objetivo común, dentro de su organización. Dentro de esta arquitectura, se han creado cuatro organizaciones diferentes que cubren los distintos aspectos del sistema: una que se encarga de la *comunicación con el exterior*, otra que estructura y determina los mecanismos *de comunicación interna del sistema* y las otras dos organizaciones internas encargadas del *razonamiento y de la generación de las soluciones* una a los distintos problemas a los que se puede enfrentar esta arquitectura, una y, la otra, encargada de los *servicios adicionales.*

OBaMADE ha sido aplicada a dos *casos de estudio* que se explicarán también dentro de este último apéndice: en primer lugar se ha utilizado para *predecir la evolución de las mareas negras* y, posteriormente, se aplicó a la *predicción de la evolución de los incendios forestales*. En ambos casos los resultados han sido satisfactorios, mostrándose eficiente a la hora de generar predicciones sobre áreas geográficas concretas y basándose siempre en datos históricos almacenados en el sistema.

D.1. OBJETIVOS FUNDAMENTALES

El objetivo principal de este trabajo de investigación es *desarrollar una arquitectura que permita resolver los problemas relacionados con los entornos distribuidos.* Para lograr ese objetivo se ha creado una *arquitectura multiagente basada en organizaciones de agentes*. Dichos agentes, estructurados en organizaciones, ofrecen distintas interfaces a los usuarios

dependiendo del tipo de dispositivo desde el que se acceda a los sistemas creados bajo esta arquitectura. Los agentes que forman parte de las organizaciones internas de la arquitectura, aquellas encargadas de generar las soluciones a los problemas planteados, siguen una metodología de *razonamiento basado en casos.* La citada metodología se basa en la reutilización de información pasada, utilizando las soluciones dadas a problemas pasados, para solucionar nuevos problemas similares a aquellos que han sido previamente solucionados y cuya solución está almacenada en el sistema relacionada con el problema.

Además del objetivo principal anteriormente citado, este trabajo de investigación se plantea cubrir otra serie de objetivos relacionados directa e indirectamente con la consecución de dicho objetivo principal, los cuales se enumeran a continuación:

- Realizar un *completo estudio y estado del arte* de las distintas técnicas y metodologías aplicadas a la solución de problemas en entornos distribuidos.
- Estudiar las *distintas metodologías y sistemas* tanto de agentes, como multiagentes y de organizaciones de agentes, para poder elegir el más apropiado para los requisitos necesitados por la arquitectura que se desarrolla en esta investigación.
- Aplicar la teoría de *organizaciones de agentes* a la creación de una arquitectura para la solución de problemas de entornos distribuidos.
- *Comparar*, de forma teórica, las ventajas y desventajas de las distintas *alternativas* a OBaMADE.
- Aplicar la arquitectura propuesta a distintos *casos de estudio* para evaluar, empíricamente, los resultados de la aplicación de la arquitectura a situaciones reales.

D.2. ENTORNOS DISTRIBUIDOS

En este documento y en toda la investigación aquí recogida, se entiende por *entorno distribuido aquel en el que los distintos componentes que interaccionan con un sistema no tienen por qué estar localizados en un mismo lugar ni a la vez.*

Las principales características de los entornos distribuidos son:

- Debe existir una *separación funcional* entre los distintos componentes que forman el sistema, permitiendo habilitar mecanismos específicos para cada una de las distintas partes, así como dotando de independencia real a cada elemento individual.
- Las distintas entidades que forman parte de los sistemas están *distribuidas de forma inherente.* Cada elemento debe funcionar dentro del sistema sin tener por qué conocer la existencia de otros elementos en el mismo.
- Los sistemas deben ser *confiables*. Los datos deben estar seguros y, a ser posible, replicados en varias localizaciones.
- Estos sistemas deben ser también *escalables,* pudiendo incorporar nuevas aplicaciones sin menoscabo de las existentes previamente.
- El hecho de compartir recursos hace que el sistema global resulte más *económico* que disponiendo de recursos individuales para cada elemento del sistema.

D.2.1. CARACTERÍSTICAS FUNDAMENTALES

Las características más importantes de los entornos distribuidos son las que se explican a continuación:

- *Heterogeneidad de los componentes.* La interconexión, sobre todo cuando se usa Internet, se da sobre una gran variedad de elementos hardware y software, por lo cual se necesitan ciertos estándares que

permitan esta comunicación. Los middleware, son elementos software que permiten una abstracción de la programación y el enmascaramiento de la heterogeneidad subyacente sobre las redes. También el middleware proporciona un modelo computacional uniforme.

- *Extensibilidad.* Determina si el sistema puede crecer y ser reimplementado en diversos aspectos (añadir y quitar componentes). La integración de componentes escritos por diferentes programadores es un auténtico reto.
- *Seguridad.* Reviste gran importancia por el valor intrínseco para los usuarios. Tiene tres componentes:
    - *Confidencialidad.* Protección contra individuos no autorizados.
    - *Integridad.* Protección contra la alteración o corrupción.
    - *Disponibilidad.* Protección contra la interferencia con los procedimientos de acceso a los recursos.
- *Escalabilidad.* El sistema es escalable si conserva su efectividad al ocurrir un incremento considerable en el número de recursos y en el número de usuarios.
- *Tratamiento de Fallos.* Consiste en la posibilidad que tiene el sistema para seguir funcionando tras producirse fallos de algún componente en forma independiente, pero para esto se tiene que tener alguna alternativa de solución. Las técnicas existentes para tratar estos fallos son las siguientes:
    - *Detección de fallos.* Algunos fallos son detectables, con comprobaciones rutinarias realizadas por el sistema, en las que se comprueba el correcto funcionamiento de los distintos elementos.

- *Enmascaramiento de fallos.* Algunos fallos detectados pueden ocultarse o atenuarse reduciendo, en lo posible, la repercusión de los mismos.
- *Tolerancia de fallos.* Sobre todo en Internet se dan muchos fallos y no es muy conveniente ocultarlos, es mejor tolerarlos y continuar. El resultado final no va a variar sustancialmente si se empleara otra técnica. Ej.: Tiempo de vida de una búsqueda.
- *Recuperación frente a fallos.* Tras un fallo se deberá tener la capacidad de volver a un estado anterior estable y sin fallos.
- *Redundancia.* Se puede usar para tolerar ciertos fallos (DNS, BD, etc.)

– *Concurrencia.* Consiste en compartir recursos por parte de varios clientes a la vez.

– *Transparencia.* Es la ocultación al usuario y al programador de aplicaciones de la separación de los componentes en un sistema distribuido. Se identifican ocho formas de transparencia:

- *De Acceso.* Se accede a recursos locales y remotos de forma idéntica.
- *De ubicación.* Permite acceder a los recursos sin conocer su ubicación.
- *De concurrencia.* Usar un recurso compartido sin interferencia.
- *De replicación.* Ofrece la posibilidad utilizar varios ejemplares de cada recurso, aumentando el rendimiento global del sistema.
- *Frente a fallos.* Logra ocultar los fallos ante los usuarios.
- *De movilidad.* Permite la reubicación de recursos y clientes sin afectar al sistema.

- *De prestaciones*. Posibilita la reconfiguración del sistema para mejorar las prestaciones según su carga.
- *De escalado.* Permite al sistema y a las aplicaciones crecer sin modificar la estructura del sistema o los algoritmos de aplicación.

D.2.2. VENTAJAS Y DESVENTAJAS

Los entornos distribuidos tienen las siguientes ventajas comparados con los *sistemas centralizados*:

- Una de las ventajas de los sistemas distribuidos es la *economía*, pues es mucho más barato añadir servidores y clientes cuando se requiere aumentar la potencia de procesamiento.
- *El trabajo en equipo*. Por ejemplo: en una fábrica de ensamblado, los robots tienen sus CPUs diferentes y realizan acciones en conjunto, dirigidos por un sistema distribuido.
- *La mayor confiabilidad.* Al estar distribuida la carga de trabajo en muchas máquinas el fallo de una de ellas no afecta tanto a las demás, el sistema sobrevive como un todo.
- *La capacidad de crecimiento incremental.* Se pueden añadir elementos de procesamiento al sistema incrementando su potencia en forma gradual según sus necesidades.

Por otro lado, este tipo de sistemas también tiene una serie de desventajas, citadas a continuación:

- El principal problema es el software, ya que el diseño, implantación y uso del software distribuido presenta numerosos inconvenientes.
- También plantea interrogantes como el tipo de S.O., programación o aplicaciones más adecuados para este tipo de sistemas, la cantidad de información que debe estar disponible para los usuarios y el reparto de tareas entre los usuarios y los sistemas.

- Las redes de comunicación también pueden representar un problema para este tipo de sistemas. Por ejemplo: pérdida de mensajes, saturación en el tráfico, etc.
- El uso compartido de datos también representa un potencial problema para estos sistemas, al tener que considerar, de forma constante, la seguridad y estabilidad de los mismos.

En general, y especialmente al tener en cuenta la aplicabilidad de estos sistemas, se considera que las ventajas superan a las desventajas, si estas últimas se administran seriamente.

D.3. AGENTES, SISTEMAS MULTIAGENTE Y ORGANIZACIONES

En los desarrollos iniciales de sistemas multiagente, los diseñadores se centraron en el estudio del agente, es decir, en la estructura interna del mismo y en su comportamiento. Las organizaciones, como mucho, emergían de las interacciones de los agentes [Boissier *et al.*, 2007], por ejemplo con los protocolos de tipo ContractNet o la formación de coaliciones de dependencia. Sin embargo, los métodos de análisis y diseño de sistemas multiagente no consideraban a las organizaciones como entidades propias, ni tampoco los agentes las trataban como conceptos sobre los que razonar. En realidad, los agentes eran vistos como entidades autónomas y dinámicas que evolucionaban en función de sus propios objetivos, sin que existieran restricciones explícitas externas sobre sus comportamientos ni comunicaciones [Boissier *et al.*, 2007].

El concepto de agente tiene su principal origen en la inteligencia artificial, evolucionando como una entidad computacional aislada gracias a la influencia de la ingeniería de software, superando así las limitaciones de las metodologías orientadas a objetos. La principal diferencia entre los

conceptos de agente y de objeto es la autonomía que poseen los primeros. Los agentes son capaces de tomar decisiones, reaccionar ante estímulos externos, cambiar su propio comportamiento y adaptarse a las necesidades del entorno.

La definición del término *agente* es todavía tema de discusión, ya que se asocia a un gran número de disciplinas, desde la psicología, hasta las orientadas a la computación, tales como la inteligencia artificial, la ingeniería de software y las bases de datos, entre otras, por lo que se hace difícil realizar una definición con una visión global independiente del área de influencia. Wooldridge define un agente como *un sistema computacional que se sitúa en algún entorno y es capaz de actuar de forma autónoma en dicho entorno para alcanzar sus objetivos de diseño* [Wooldridge, 2002]. En cambio, Russell, et al. [Russell *et al.*, 1995] consideran que la noción de un agente aparece como *una herramienta para analizar sistemas*, no una caracterización absoluta que divida el mundo en agentes y no agentes. Para este último autor, un agente es *cualquier elemento capaz de percibir su entorno a través de sensores y responder según su función en el mismo entorno a través de actuadores*, asumiendo que cada agente puede percibir sus propias acciones y aprender de la experiencia para definir su comportamiento.

Debido a que existen grandes diferencias y discusión a la hora de definir lo que es un agente, se ha optado por definir una serie de características que éstos deben cumplir:

- *Autonomía.* Actuar sin la necesidad de intervenciones externas, ya sean humanos u otros agentes, y tener alguna clase de control sobre sus acciones y su estado interno.
- *Situación.* Situarse dentro de un entorno, ya sea real o virtual.
- *Reactividad.* Percibir su entorno y actuar sobre éste con la capacidad de adaptarse a sus necesidades.

- *Pro-Actividad o Racionalidad.* Tomar la iniciativa para definir metas y planes que les permitan alcanzar sus objetivos.
- *Habilidad social.* Interactuar con otros agentes, incluso con humanos.
- *Inteligencia.* Rodearse de conocimiento (creencias, deseos, intenciones y metas).
- *Organización.* Capacidad de agruparse dentro de sociedades que siguen unas estructuras similares a las definidas en sociedades humanas o ecológicas.
- *Aprendizaje.* Habilidad de adaptarse progresivamente a cambios en entornos dinámicos, mediante técnicas de aprendizaje.

Una vez descritos los principales requisitos que debe cumplir un agente y las características de los diferentes tipos de agentes que existen, es necesario definir lo que es un sistema multiagente (MAS: Multi-Agent System). Un *sistema multiagente* es básicamente *una red de entidades enfocadas a resolver problemas, y que trabajan de manera conjunta para encontrar respuestas a los problemas que están más allá de las capacidades o del conocimiento individuales de cada entidad* [Durfee *et al.*, 1989].

Una definición más general y actualizada describe un sistema multiagente como *cualquier sistema compuesto de múltiples componentes autónomos que presentan las siguientes características* [Jennings *et al.*, 1998]:

- Cada agente tiene capacidades incompletas para resolver un problema.
- No existe un sistema de control global.
- Los datos son descentralizados.
- La computación es asíncrona.

D.3.1. SISTEMAS MULTIAGENTE

Las arquitecturas para la construcción de agentes especifican cómo se descomponen los agentes en un conjunto de módulos que interactúan entre sí para lograr la funcionalidad requerida. Entre las principales tenemos las siguientes, diferenciadas en el modelo de razonamiento que utilizan:

- *Reactivas.* Carecen de razonamiento simbólico complejo y de conocimiento o representación de su entorno, por lo que sus mecanismos de comunicación con otros agentes son muy básicos. Los agentes que utilizan este tipo de arquitectura reciben estímulos de su entorno y reaccionan ante ellos modificando sus comportamientos y el mismo entorno.
- *Deliberativas.* Utilizan modelos de representación simbólica del conocimiento basados en la planificación. Los agentes deliberativos emplean mecanismos de comunicación complejos y contienen un modelo simbólico del entorno. Toman decisiones utilizando razonamiento lógico basado en la concordancia de patrones y en la manipulación simbólica, partiendo de un estado inicial y un conjunto de planes con un objetivo a satisfacer.
- *Híbridas.* Son arquitecturas intermedias entre las dos anteriores. Los agentes de este tipo incluyen comportamientos reactivos y deliberativos, generando un ciclo *percepción-decisión-acción*. El comportamiento reactivo se utiliza para reaccionar ante eventos que no requieran decisiones complejas sobre ciertas acciones.

Cada tipo de agente cuenta con características distintas para cada escenario de aplicación en el que se desenvuelva. Por ejemplo, en un entorno rodeado de sensores y en el cual el tiempo de reacción ante los estímulos sea lo más importante, los agentes reactivos son la opción más

recomendable. Sin embargo, en ciertos escenarios puede ser necesario que los agentes sean capaces de tomar decisiones más complejas y de forma dinámica, por lo que el uso de agentes deliberativos o híbridos resulta más conveniente.

Como los datos se encuentran organizados de forma distribuida y no existe un sistema de control global. Cada agente se centra en su propia conducta, tomando la iniciativa guiado por sus objetivos y decidiendo dinámicamente las tareas que debe realizar o asignar a otros agentes. Es necesario que los agentes trabajen de forma coordinada, principalmente a través de mecanismos de negociación, para alcanzar sus objetivos [Ossowski and García-Serrano, 1998].

Las características de los agentes deliberativos *BDI (Belief, Desire, Intention)*, así como la posibilidad para modelar sus capacidades e integrar mecanismos de razonamiento, hacen que resulten adecuados para la resolución de problemas en tiempo de ejecución en entornos altamente dinámicos. Como consecuencia, los agentes permiten a los sistemas aprender de las experiencias pasadas y reaccionar de manera diferente de acuerdo a las necesidades de los usuarios y las características del contexto en una situación determinada, requerimientos fundamentales para afrontar los retos que plantean los entornos distribuidos. Por su parte, la combinación de las herramientas para la ingeniería del software Gaia y SysML, permiten obtener modelos de los sistemas multiagente cercanos a la implementación, facilitando la labor de los desarrolladores.

Sin lugar a dudas, los sistemas multiagente representan una interesante alternativa que bien vale la pena explorar para intentar afrontar los retos que presenta los entornos distribuidos, especialmente en el desarrollo de sistemas dinámicos y adaptables a las necesidades de los usuarios.

D.3.2. METODOLOGÍAS MULTIAGENTE ORIENTADAS A LAS ORGANIZACIONES

En este tipo de metodologías el diseñador del MAS se centra desde un principio en la organización del sistema. Por tanto, analiza el MAS desde una perspectiva global, de modo que el proceso de desarrollo se guía por los conceptos organizativos [Argente *et al.*, 2006].

Estos métodos aparecen como consecuencia de la necesidad de diseñar sistemas que permitan tener en consideración aspectos como la estructura de la organización, sus objetivos, sus normas, etc. desde las etapas iniciales del desarrollo del sistema.

Los objetivos de la organización representan una descripción a alto nivel de los propósitos de la sociedad. Permiten guiar las decisiones sobre cómo se debe diseñar la estructura de la organización. Así, los objetivos determinan las tareas que se deben llevar a cabo, el tipo de agentes y sus habilidades requeridas, y el reparto de los recursos entre los miembros de la organización.

La estructura de la organización queda formalizada cuando los principios que gobiernan su comportamiento se formulan de forma precisa. Los roles y sus relaciones se definen de forma independiente de los atributos y dependencias de las personas o agentes que ocupen una posición particular en la estructura de la organización.

Por tanto, dicha estructura viene descrita por los roles, sus interacciones y el lenguaje de comunicación que empleen. Los roles representan las diferentes entidades o actividades necesarias para cumplir con el propósito de la organización. Además, los objetivos globales de la sociedad conforman el punto de partida para especificar los objetivos y acciones a asignar a los roles.

Finalmente, las normas sociales describen el comportamiento esperado de los miembros (desde el punto de vista del diseño de la organización) y las sanciones que se deben aplicar en el caso de realizar acciones no deseables. Las normas suelen ser establecidas y ejecutadas por instituciones que tienen un estatus legal y, por tanto, conceden legitimidad y seguridad a los miembros de la sociedad.

Tras el estudio de distintos trabajos que siguen esta perspectiva metodológica, se observan dos tendencias bien diferenciadas. Por un lado, algunas metodologías se centran solamente en la estructura organizativa, sin realizar de forma explícita el análisis y diseño de las normas sociales. Ejemplos de estas metodologías son Agent-Group-Role [Ferber *et al.*, 2004], Roadmap [Juan *et al.*, 2002], la extensión de Tropos [Kolp *et al.*, 2003], MESSAGE [Caire *et al.*, 2002], INGENIAS [Sanz, 2002], ANEMONA [Boggino, 2005, Giret B., 2005] o AML [Cervenka and Trencansky, 2007].

Por otro lado, otras metodologías se centran en las normas sociales y definen de forma explícita mecanismos de control para establecer las normas y controlar su ejecución. Además, estas metodologías consideran ciertos mecanismos para incluir agentes externos en la sociedad y controlar su comportamiento. Por tanto, resultan adecuadas para el diseño de sistemas multiagente abiertos. Ejemplos de este tipo de metodologías son OperA [Dignum, 2004], Civil Agent Societies [Dellarocas and Klein, 2000b], SODA [Omicini, 2001], MOISE [Gateau *et al.*, 2005] y la extensión de Gaia [Zambonelli *et al.*, 2003]. Además, el marco de trabajo Electronic Institutions [Esteva *et al.*, 2001] se centra en la perspectiva organizativa y el control de las normas sociales. Así mismo, el marco de trabajo HARMONIA [Vázquez-Salceda and Dignum, 2003] permite modelar las normas de las organizaciones electrónicas en varios niveles, desde el más abstracto, tomando como base los estatutos de la

organización, hasta el nivel procedimental en el que se implementan los procedimientos y protocolos finales de las normas. Posteriormente, este marco de trabajo se unió a la metodología OperA, definiendo así un nuevo método denominado OMNI [Vázquez-Salceda *et al.*, 2005].

D4. ARQUITECTURA BASADA EN ORGANIZACIONES PARA ENTORNOS DISTRIBUIDOS

OBaMADE, la arquitectura presentada en este documento, representa una combinación de técnicas y metodologías adaptadas a entornos distribuidos que la hacen aplicable a distintos tipos de situaciones.

D.4.1. ELEMENTOS FUNDAMENTALES

OBaMADE es una arquitectura basada en organizaciones de agentes. Dichas estructuras potencian los elementos *sociales* de los agentes, dando importancia a su colaboración para lograr un objetivo común.

OBaMADE está compuesta por cuatro organizaciones fundamentales. Dichas organizaciones están representadas de forma esquemática en la *figura 47*. En primer lugar está la *Organización de Interfaces*, que se encarga de la comunicación con el exterior. Esta organización presenta las distintas interfaces a los usuarios dependiendo tanto del tipo de servicio que soliciten como del dispositivo que estén usando. Tanto lo uno como lo otro serán posteriormente transparentes para el resto de elementos del sistema, que simplemente se encargarán de solucionar las solicitudes que, desde esta organización, se vayan generando.

La *Organización de Comunicación* es la encargada de recibir las solicitudes que se generan en la *organización de interfaces*. Esta organización gestiona el tipo de solicitud que recibe el sistema y la envía a la correspondiente organización de servicios que tenga que dar solución a dicha solicitud. Es en esta organización donde se determina qué servicios están disponibles y qué servicios pueden solucionar los distintos tipos de solicitudes admitidas por el sistema.

Por último hay dos organizaciones de servicios. Se trata de la *Organización de Servicios CBR* y la *Organización de Servicios Adicionales*. Dichas organizaciones agrupan a los distintos agentes que dan *servicio* a las distintas solicitudes venidas desde los usuarios.

D.4.2. RAZONAMIENTO BASADO EN CASOS

OBaMADE utiliza la metodología de *Razonamiento Basado en Casos* (conocida por *CBR, Case-Based Reasoning*), como base para crear sus procesos de razonamiento interno. Basándose en ella se han desarrollado unos servicios implementados por agentes que forman

parte de la *Organización de Servicios CBR* que genera las distintas soluciones de las diferentes aplicaciones de esta arquitectura. Por eso, en esta sección se explican los fundamentos de esta metodología.

El *Razonamiento Basado en Casos* es un método comúnmente utilizado para solucionar nuevos problemas basándose en las soluciones de problemas anteriores. Un mecánico de automóviles que repara un motor porque recordó que otro vehículo presentaba los mismos síntomas está usando razonamiento basado en casos. Un abogado que apela a precedentes legales para defender alguna causa está también utilizando este tipo de razonamiento basado en casos. Cuando un ingeniero copia elementos de la naturaleza, está tratando a ésta como una *"base de datos de soluciones"*. El razonamiento basado en casos es una manera de razonar haciendo analogías. Se ha argumentado que más que un método poderoso para el razonamiento de computadoras, es un sistema usado por las personas para solucionar problemas cotidianos. Más radicalmente se ha sostenido que todo razonamiento es basado en casos, porque está basado en la experiencia previa.

Podemos definir claramente el razonamiento basado en casos partiendo de una definición clásica de esta metodología:

"*A case is a contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner*", [Kolodner, 1993]

Este sistema de razonamiento se basa en una unidad mínima llamada caso, como literalmente define Kolodner. Un caso se puede definir como una representación de una experiencia anterior, una vivencia. Podría ser visto como una caja en la que encontramos todas aquellas cosas que ocurrieron y de las que se saben causas y consecuencias.

El *'case'* del que se habla en la definición original está modificado por *'contextualizad piece of knowledge'*. Es importante hacer notar sobre todo el término *contextualizado* ya que, como se ha indicado anteriormente este conocimiento representa un conjunto de hechos que han transcurrido en la experiencia. Una parte de estos hechos corresponden al contexto en el que transcurre la experiencia. Este contexto en el sistema experto también tiene mucha importancia ya que puede ser utilizado en el proceso de inferencia; esto se explicará más adelante.

Otro elemento importante de la definición es: *'representing an experience'*, que implica que el caso está basado en un conocimiento, es decir, no es algo creado artificialmente sobre hechos sino que está basado en un conocimiento existente previamente y, por lo tanto, que podemos considerarlo cierto desde el inicio. Además, el hecho de que se hable de experiencia comienza a hacer notar que este sistema estará muy ligado a la adquisición de conocimiento externo ya que, al estar basado en las experiencias, será necesario que el sistema vaya adquiriendo nuevas experiencias para mejorar su razonamiento.

Si se continúa con la definición, lo siguiente es: *'that theaches a lesson fundamental'*. Con esto lo que se quiere indicar es que las experiencias que hay en el sistema no se refieren a cualquier experiencia, sino sólo a aquellas que aportan alguna información sobre el tema tratado por el sistema, además de no repetir experiencias ya existentes con el mismo contexto o que no aportan nueva información al sistema. Finalmente la definición acaba con *'to achieving the goals of the reasoner'* que indica que el uso de los casos persigue directamente la consecución de los objetivos del razonamiento.

El ciclo principal que conforma el razonamiento basado en casos puede dividirse en cuatro subprocesos diferentes que se

muestran gráficamente en la *figura 48*:

- ***Recuperar*** los casos similares al que analizamos.
- ***Reutilizar*** la información y el conocimiento que tenemos en este caso para resolver el problema.
- ***Revisar*** la solución propuesta.
- ***Retener*** las partes de esta experiencia que nos puedan ser útiles para la resolución de futuros problemas.

Figura 48. *Ciclo básico del Razonamiento Basado en Casos.*

Cuando un nuevo problema llega a un sistema primero que hay que hacer es dado ese determinado problema *recuperar* los casos relevantes que pueden solucionarlo.

Una vez se tiene este conjunto de casos que guardan cierta similitud con el caso para el cual hay que proponer una solución hay que *reutilizar* la solución de todos ellos, en su globalidad o solamente

en alguna de sus partes que interese para transformar sus contextos en el problema que se tiene actualmente. Con ello se tendría una primera versión de la solución que es necesario probar en el mundo real o en una simulación y es preciso *revisarla*. Se trata de un proceso circular en el que reutilizan diversos casos de la base de conocimiento, se revisa la solución y, si no es satisfactoria, se vuelve a modificar con la eliminación de los casos que fuesen incorrectos o la inclusión de aquellos que faltasen para perfeccionar la solución.

Finalmente el último paso es la *retención*. Después de que la solución haya sido adaptada satisfactoriamente para resolver el problema dado, se almacena la experiencia resultante como un nuevo caso en la memoria. Uno de los objetivos del razonamiento basado en casos reside no solo en recordar los casos resultantes que hayan sido acertados, sino también, aquellos en que se ha fallado, ya que con estos se puede mejorar el razonamiento del sistema para que cuando se tenga que llevar a cabo un proceso similar se sepa que no hay que seguir esa línea de razonamiento que lleva a un resultado incorrecto.

D.4.3. CAMPOS DE APLICACIÓN DE OBAMADE

OBaMADE se ha desarrollado de forma genérica, sin estar directamente relacionada con un tipo de problema específico. Sus características hacen que pueda ser aplicada en diferentes tipos de situaciones. Los distintos elementos que forman parte de ella, le permiten ofrecer servicios de comunicación entre distintos usuarios y la estructura interna que contiene la información. Dicha comunicación permite que se pueda adaptar a distintos tipos de problemas.

Así, los tres principales campos de aplicación de esta arquitectura son: la *generación de predicciones*, la *clasificación y agrupamiento* y la *planificación.* Los tres serán explicados a continuación.

La principal y primera aplicación en la que puede utilizarse OBaMADE es la *generación de predicciones.* Para ello, el sistema almacena información con parámetros temporales, que caracterizan una situación en un momento y en el momento siguiente, representando así la evolución temporal de un determinado entorno. De esta manera, analizando casos almacenados en el sistema que tuvieran un estado de partida similar a aquel del que queremos obtener la predicción, podremos generar una predicción fiable.

La información se inserta en el sistema desde diferentes fuentes, bien sean usuarios que quieren ampliarlo sin necesidad de pedir una predicción, satélites con información en tiempo real, sensores o bases de datos accesibles por el sistema. Toda esta información se estructura y organiza dentro de la base de casos para poder ser utilizada a la hora de generar futuras predicciones.

La *clasificación* consiste en estructurar la información en un cierto número de categorías dependiendo de las características intrínsecas de dicha información. El *agrupamiento* (normalmente conocido por su correspondiente anglicismo: *clustering*), consiste en determinar los posibles grupos diferentes en que se distribuyen una serie de elementos dados. Estas dos técnicas están muy relacionadas y OBaMADE puede ser fácilmente utilizada en su resolución y es capaz de combinarlas para generar complejas aplicaciones de, por ejemplo, minería de datos o extracción de conocimiento. Cuando se afrontan este tipo de tareas, la fase de creación de la base de casos es fundamental, ya que es en ella donde se van a determinar las categorías. Bien sea para clasificar o para agrupar, es en esta fase donde se analiza la información disponible y se crean y organizan las distintas categorías. Una vez hecho este trabajo, cualquier clasificación o agrupamiento posterior estará basado en la información almacenada en la base de casos y seguirá la misma organización.

Un último campo en el que puede ser aplicada esta arquitectura es el de *planificación*. En este caso, la metodología seguida por los servicios internos de la arquitectura no será de *razonamiento basado en casos*, sino de *planificación basada en casos*. Los métodos de funcionamiento son similares, ya que los planes almacenados en la base de casos se agrupan en función de las condiciones para las que se generaron dichos planes. Los planificadores creados bajo esta arquitectura no pueden ser de tipo general, sino siempre aplicados a algún campo de conocimiento determinado, que establecerá las relaciones entre las causas o situaciones iniciales y las consecuencias o soluciones a dichas situaciones.

D5. RESULTADOS

La arquitectura OBaMADE ha sido aplicada a dos casos de estudio para validar su corrección. En primer lugar, se ha usado en la generación de predicciones respecto a la evolución de los vertidos generados tras una *marea negra*. En este caso, el sistema creado sobre OBaMADE predice la probabilidad de encontrar restos del vertido en una determinada zona del océano.

El segundo caso de estudio al que se ha aplicado OBaMADE es la predicción de la evolución de *incendios forestales*. El sistema predice, en este caso, la presencia o no de fuego en una determinada área geográfica una vez se ha declarado un incendio en las inmediaciones.

D.5.1. MAREAS NEGRAS

Cuando se produce un vertido generalizado de algún tipo de hidrocarburo en el mar (fenómeno normalmente conocido como *mareas negras*), es importante disponer de toda la información necesaria para evitar o minimizar, en la medida de lo posible, el eventual daño

medioambiental asociado a dicho vertido.

Para analizar dichos daños medioambientales es muy importante saber si una zona se va a ver afectada por los vertidos. Predecir, con suficiente antelación, este dato, puede ser de vital importancia a la hora de preservar determinadas zonas especialmente delicadas, bien en términos socio-económicos (aquellas con importantes núcleos de población o con industrias relacionadas directamente con el mar) o medioambientales (las de especial importancia por su diversidad y en buen estado de conservación).

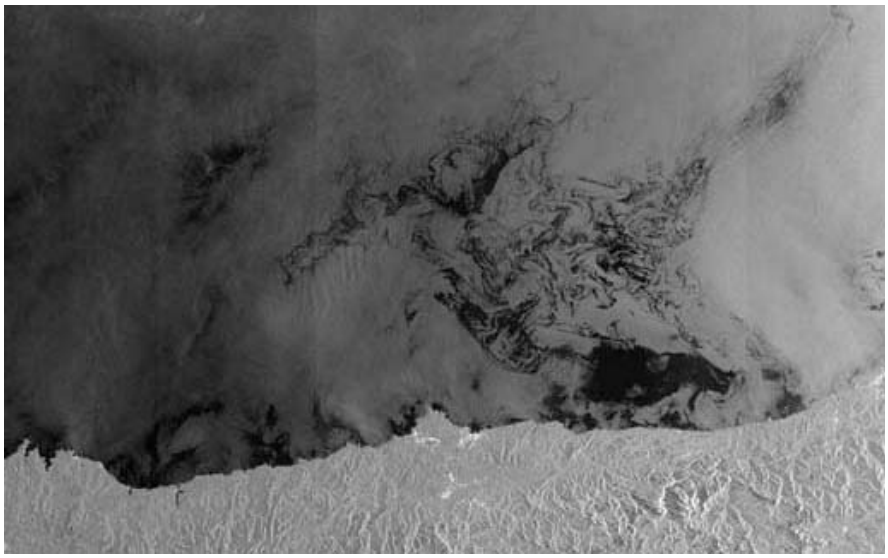


Figura 49. *Imagen de satélite de manchas originadas en el accidente del Prestige.*

OBaMADE, ha sido aplicada para generar predicciones en este caso de estudio en concreto. Para ello, se disponía de los datos históricos del accidente del petrolero Prestige, ocurrido en noviembre de 2002 cerca de las costas gallegas (en el noroeste de la Península Ibérica). La *figura 49*, muestra una imagen de satélite de una de las zonas afectadas, al norte de la Península Ibérica, en la que pueden apreciarse claramente las manchas de fuel. Dicha imagen fue obtenida días después del barco.

Los datos disponibles tienen distintos orígenes. Por un lado están las imágenes de satélite en las que se pueden ver las manchas de fuel. Dichas manchas se asocian con información meteorológica y marítima, que es obtenida de servicios de información obtenidos de los satélites, que proporcionan, en tiempo real, información referente a la meteorología (presión atmosférica, temperatura…) y al océano (oleaje, salinidad…). Toda esa información se estructuró y se almacena en la base de casos de tal forma que se establecen relaciones temporales entre las situaciones almacenadas en la base de casos., en la base de casos, se establece una relación entre la situación presente (*problema*) y la situación en el momento inmediatamente posterior (*solución*).

Cuando una solicitud de predicción entra en el sistema, lo hace a través de la *Organización de Interfaces* que, como ya se ha explicado con anterioridad, es la encargada de proporcionar a cada usuario el interfaz que necesita para interactuar con la aplicación dependiendo del tipo de dispositivo que esté manejando y, también, del tipo de servicio que vaya a demandar.

Tras pasar por la *Organización de Interfaces*, la solicitud llega a la *Organización de Comunicación*, que la analiza para, a su vez, pasársela a la *Organización de Servicios* correspondiente, bien sea la relativa a *servicios CBR* o la encargada de los *servicios adicionales.*

Si, como es el caso, se trata de una solicitud de predicción, dicha solicitud llegará a la *Organización de Servicios CBR*, que será la encargada de, mediante los correspondientes agentes encargados de las distintas fases del ciclo CBR, generar la predicción para una situación en concreto. Para realizar la predicción, el usuario debe introducir el área geográfica de la que quiere conocer la predicción y los datos de los que disponga, especialmente su localización y tamaño si visualiza, de forma directa, alguna mancha de fuel.

Para completar los datos necesarios para generar los *casos*, el sistema accederá a datos de satélites que proporcionan las variables meteorológicas y oceánicas necesarias. Así, se completarán todos los parámetros que se van a tener en cuenta: longitud, latitud, fecha, oleaje, presión atmosférica, salinidad, temperatura del mar, área de las manchas, -dirección y fuerza del viento, dirección y fuerza de la corriente marítima. El área a analizar se divide en pequeñas regiones cuadradas, que son las que delimitan los *casos.* Para cada una de esas regiones se almacenan todas las variables anteriormente citadas. El parámetro denominado *área de las manchas* se refiere a la proporción de la zona que está ocupada por manchas. Ese parámetro es sobre el que se realiza la predicción, obteniendo, al final de la misma, un valor futuro de ese parámetro.

Para realizar la predicción, se extraen de la base de casos un conjunto de casos que sean similares al problema introducido en el sistema. La base de casos está organizada de tal forma que, aquellos casos que sean parecidos se almacenarán próximos unos a los otros. De esta forma, resulta más sencillo y rápido recuperar de ella un grupo de casos parecidos.

Con el grupo de casos recuperados se genera la predicción, utilizando una red neuronal GRBF entrenada al efecto. Dicha red proporcionará, como salida, un valor futuro para el parámetro *área de las manchas* de cada una de las regiones cuadradas que se le pasen.

Para validar la aplicación, se han comparado los resultados obtenidos con OBaMADE con otras técnicas. La *figura 50* muestra una representación gráfica de los resultados obtenidos en la citada comparación. En dicha figura se pueden ver la evolución de los resultados a medida que el tamaño de la base de casos ha ido creciendo. Cuando el número de casos almacenado se incrementa, los resultados van mejorando de forma progresiva. Esto resulta lógico ya que, al aumentar la

variabilidad de casos almacenados y su número, la posibilidad de encontrar casos parecidos al que se quiere resolver aumenta y los resultados mejorarán.

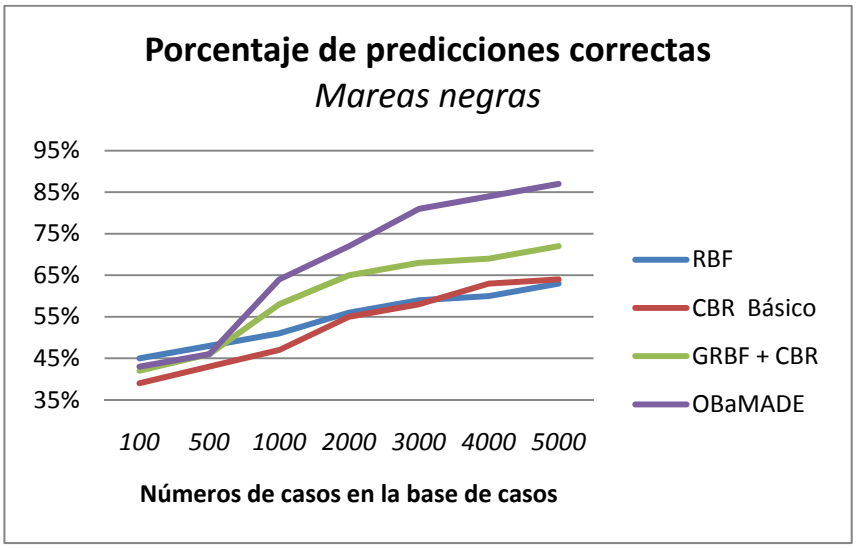


Figura 50. *Porcentaje de predicciones correctas tras aplicar OBaMADE al problema de las mareas negras.*

En la *figura 50* se compara la arquitectura OBaMADE con otras técnicas. En primer lugar se comprobaron los resultados de realizar predicciones con una red neuronal *RBF* sin ninguna otra técnica asociada. En ese caso, las predicciones se obtenían tras haber entrenado la red neuronal con los datos disponibles y, por lo tanto, los resultados no eran suficientemente satisfactorios. En segundo lugar, se aplicó un sistema *CBR Básico*, en el que los datos se almacenan en una base de casos, se recuperan los más similares y no hay técnicas adiconales aplicadas. En tercer lugar, se utilizó una combinación de *RBF y CBR*, en el que la red era sólo entrenada con aquellos casos más similares y, por lo tanto, los resultados mejoraban. Por último, se aplicó el sistema creado sobre la arquitectura *OBaMADE*, generando los mejores resultados de entre los sistemas comparados.

D.5.2. INCENDIOS FORESTALES

En segundo lugar, la arquitectura OBaMADE ha sido aplicada a la predicción de la evolución de incendios forestales. El funcionamiento del sistema creado sobre la arquitectura propuesta es similar al descrito en la aplicación de la arquitectura al problema de las mareas negras. En este caso, las variables almacenadas en el sistema son las siguientes: longitud, latitud, fecha, presión atmosférica, temperatura, área de los fuegos y dirección y fuerza del viento.

Figura 51. *Imagen de los experimentos llevados a cabo en Gestosa, Portugal.*

En este caso los datos históricos con los que se ha creado la base de casos provienen de unos experimentos realizados en Portugal, dentro del proyecto SPREAD [Spread, 2004]. Los experimentos de los que se tomaron los datos se realizaron en la zona de Gestosa, en el centro de Portugal, en la Serra de Lousa, a una altitud entre 800 y 950 m sobre el nivel del mar, entre los años 2002 y 2004 [Gestosa, 2005]. Dichos experimentos comenzaron en 1998 y se completaron en diciembre de 2004. Se intentó recoger datos experimentales sobre el comportamiento de los fuegos en distintas situaciones, para poder realizar un modelado de la

evolución de los mismos. Para mantener la seguridad mientras se realizaban los experimentos, se dividió el terreno en zonas de forma y dimensión regulares separadas por cortafuegos para limitar la expansión de los fuegos. La *figura 51* muestra una imagen de los experimentos llevados a cabo, en la que pueden verse las zonas delimitadas y los fuegos originados.

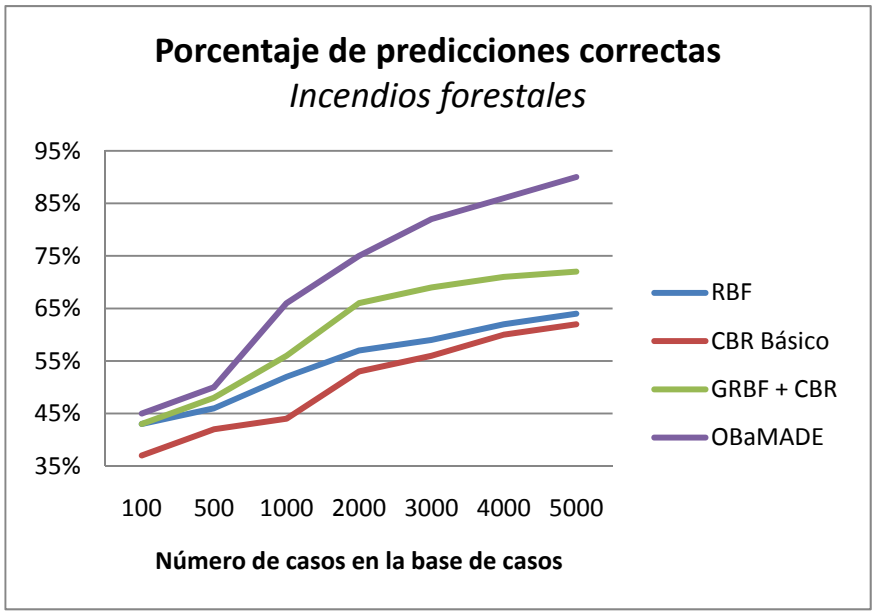


Figura 52. *Porcentaje de predicciones correctas tras aplicar OBaMADE al problema de los incendios forestales.*

Una vez se dispone de los datos en la base de casos, el funcionamiento del sistema predictivo es el mismo que el explicado en la aplicación de la arquitectura a las mareas negras. Los resultados de aplicar las distintas técnicas utilizadas para comparar el rendimiento de la arquitectura OBaMADE pueden verse en la *figura 52*. Al igual que sucedió con el caso de las mareas negras, los resultados mejoran a medida que aumenta la cantidad de información almacenada en la base de casos.

Así mismo, también puede verse que, de las técnicas comparadas, es el sistema basado en la arquitectura OBaMADE el que mejores resultados obtiene.

Tras aplicar la arquitectura presentada en este documento a dos casos de estudio, ha podido comprobarse los positivos resultados obtenidos, siendo esperanzador para poder aplicar esta misma arquitectura a otro tipo de problemas y de campos de conocimiento en los que poder desarrollar sus capacidades de generación de soluciones a partir de datos almacenados.

D6. CONCLUSIONES Y TRABAJO FUTURO

En este documento se ha presentado una nueva arquitectura multiagente basada en organizaciones y diseñada para ser utilizada en entornos distribuidos. Dicha arquitectura, llamada OBaMADE, está formada por una serie de organizaciones de agentes que colaboran para poder obtener soluciones a los distintos problemas a los que puede ser aplicada.

La arquitectura OBaMADE proporciona un entorno de trabajo suficientemente flexible como para cubrir los requerimientos de los sistemas diseñados para solucionar problemas de entornos distribuidos. Situaciones dinámicas en las que hay gran interacción por parte de los usuarios de forma asíncrona son adecuadamente solucionadas por esta arquitectura. Sus distintos elementos funcionan de forma distribuida, colaborando para obtener un resultado común.

El uso de agentes ligeros en un entorno distribuido con capacidades comunicativas permite a los sistemas creados sobre esta arquitectura obtener una comunicación transparente para el usuario, sin tener que notificar cada intercambio comunicativo. Los usuarios obtendrán los mismos resultados independientemente de su localización de los dispositivos desde los que se acceda a los sistemas creados.

El núcleo del sistema está formado por un conjunto de servicios que siguen la metodología del razonamiento basado en casos. Dichos servicios están implementados por una serie de agentes que cubren las fases básicas del ciclo del razonamiento basado en casos. Estos integran una serie de técnicas de inteligencia artificial diseñadas para extraer el conocimiento disponible en la información almacenada. Estos agentes, como parte de una de las organizaciones de la arquitectura, pueden comunicarse entre ellos para lograr un objetivo común y tomar las mejores decisiones en cada momento.

El empleo de agentes ligeros permite, además, expandir las posibilidades de desarrollo de aplicaciones basadas en la arquitectura OBaMADE a dispositivos que no tienen por qué disponer de una alta capacidad de procesamiento (teléfonos móviles, PDAs…).

La arquitectura OBaMADE puede ser aplicada a distintos tipos de problemas, desde problemas de predicción, hasta clasificación y agrupamiento, pasando por problemas de planificación. En concreto ha sido aplicada a dos casos de estudio en los que ha demostrado su capacidad para la generación de predicciones. En ambos se ha demostrado la validez y la calidad de los resultados obtenidos por los sistemas basados en esta arquitectura. Será necesario aplicar OBaMADE a otro tipo de problemas que permitan demostrar empíricamente las ventajas que, desde el punto de vista teórico, se vislumbran en la utilización de esta arquitectura.

Las técnicas de inteligencia artificial usadas para resolver los distintos servicios ofrecidos por la arquitectura han demostrado su validez en los dos casos de estudio analizados en este trabajo. Sería interesante poder incorporar más técnicas de tal forma que presente varias opciones en los distintos servicios y permita elegir en función, por ejemplo, del tipo de problema que se va a resolver.

Aunque, como se ha explicado con anterioridad, la arquitectura se ha probado en situaciones reales, sería necesario realizar pruebas exhaustivas

para evaluar todos los detalles de la arquitectura propuesta en términos de tiempo, simplicidad y calidad del análisis y del diseño. La calidad de los resultados generados por los sistemas diseñados basándose en esta arquitectura también debe ser evaluada.

A lo largo de este documento se ha explicado OBaMADE, una nueva arquitectura basada en organizaciones de agentes diseñada para ser aplicada a entornos distribuidos. Los resultados obtenidos tras la creación de sistemas basados en dicha arquitectura y aplicados a ejemplos reales han sido muy esperanzadores. Las posibilidades de aplicación y desarrollo de la arquitectura son muchas y, basándose en los resultados obtenidos, se puede asegurar que podrá ser utilizada en otro tipo de entornos de forma exitosa.