A Multi-Agent System framework to support the decision-making in complex real-world domains

Master's Thesis

Thania Rendón Sallard

Advisor
Dr. Miquel Sànchez i Marrè
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

September, 2009

A Multi-Agent System framework to support the decision-making in complex real-world domains

Master's Thesis

Thania Rendón Sallard

Abstract

The aim of this work was to develop a framework capable of supporting the decision-making process in complex real-world domains, such as environmental, industrial or medical domains using a Multi-Agent approach with Rule-based Reasoning. The validation of the framework was done in the environmental domain, particularly in the area of river basins.

Contents

Chapter 1.
Introduction

1.1 Motivation

Traditional software systems often cannot cope with the intricacy of complex real-world domains due to the presence of uncertainty and approximate knowledge as is commonly the case in environmental, industrial, and, medical areas. This motivates the use of Artificial Intelligence (AI) techniques in order to confront such complexity and thus support the decision-making process.

In this thesis we shall examine the nature and application of various AI techniques including Multi-Agent Systems (MAS) and Rule-based Reasoning (RBR) in supporting the decision-making process both in a generic context and, in specific domains.

Our purpose therefore is to demonstrate how agents are able to manage the complexity of real-world domains and successfully deploy a MAS incorporating RBR. We believe that this supports our conclusion that an agent-based framework is capable of supporting the decision-making process.

To this end, we extended the functionality of a MAS platform to be able to incorporate reasoning abilities by means of RBR. The validation of the framework was done in the environmental domain, particularly in the management of river basins.

1.1.1 Decision making complexity in real world domains

Environmental and medical fields belong to a set of critical domains where incorrect management decisions may have disastrous social, economic and ecological consequences.

The complexity of environmental and medical problems requires the development and application of new tools capable of processing not only numerical aspects, but also experience from experts together with wide public participation, which are all needed in decision-making processes [Poch et al., 2004].

Complexity of real-world systems:

- *Inherent complexity of the systems*. These processes involve a huge amount of knowledge containing complex interactions between physical–chemical, biological, ecological, social and, economical processes. Furthermore, they are stochastic and very often spatial and temporal dependent processes.

- *Uncertainty or approximate knowledge.* These processes generate a considerable amount of qualitative information. Some of the sources of this uncertainty can be tamed

with additional data or further research. However, in other cases uncertainty is insurmountable. An example of this is the case for chaotic behaviour, or for self-organisation processes. It is also typical of socio-ecological systems, which involve several stakeholders, each with their own goals.

- *Huge quantity of data/information.* These domains tend to produce a great volume of data and information.

- *Many of the facts and principles* underlying the domain cannot be characterized precisely solely in terms of a mathematical theory or a deterministic model.

- *Heterogeneity and scale.* Because the media in which these processes take place are not homogeneous and cannot easily be characterized by measurable parameters, data are often heterogeneous. Moreover, the different scale times inherent to different measures in the process have to be properly integrated and managed.

- *Multiplicity of scales*. Environmental and medical problems have been associated traditionally with distinct *spatial* scales (i.e., local, national, global), each associated with specific *timescales*. However, interactions among these scales are becoming increasingly clear. Therefore, advocating a single perspective that encompasses everything in a system is becoming increasingly difficult and ineffective.

Environmental and medical issues must be considered in terms of complex systems. But not all the systems present the same level of complexity in terms of both the degree of uncertainty and the risk associated with decisions. If the degree of complexity is represented as a function of uncertainty, on the one hand, and the magnitude or importance of the decision, on the other hand, then three levels of complexity might distinguished [Funtowicz and Ravetz, 1993, 1999].

The first level of complexity would correspond to simple, low uncertainty systems where the issue at hand has limited scope. A single perspective and simple models would suffice to provide a satisfactory description of the system. The second level would correspond to systems with a higher uncertainty degree where simple models can no longer provide satisfactory descriptions. Acquired experience then becomes more and more important, and the need to involve experts in problem solving becomes advisable. Finally, the third level would correspond to truly complex systems, where uncertainty is not necessarily associated with a higher number of elements or relationships within the system, and where the issues at stake reflect conflicting goals.

According to [Sànchez-Marrè et al, 2008], most environmental and medical systems belonging to the second and third level cannot be only tackled with the traditional tools of mathematical modelling. To confront this complexity, a new paradigm is needed, and it requires new intellectual challenges.

1.1.2 Multi-Agent Systems

Multi-Agent Systems are based on the idea that a cooperative working environment can cope with problems which are hard to solve using the traditional centralized approach to computation. Intelligent agents are used to interact in a flexible and dynamic way to solve problems more efficiently [Mangina, 2002].

In a MAS, agents interact with each other within an underlying communication infrastructure and without a procedural control mechanism; and, the individual agents often are distributed and autonomous [Huhns and Stephens, 1999].

[Anderson and Evans, 1994] discuss the application of intelligent agents as an approach to modeling in natural resource management, stressing the need for autonomy and the ability of an agent to interact spatially and temporally with surrounding entities. They also underscore the equal importance of providing a satisfactory representation of the spatial world in which the agents are embedded.

Why are agents useful?

Agents are an approach to structuring and developing software that offers certain benefits, and that is very well suited to certain types of applications. Perhaps the single most important advantage of agents is that they reduce coupling.

Agents are autonomous, which can be seen as encapsulating invocation [Odell, 2002; Parunak, 1997]. Coupling is reduced not only by the encapsulation provided by autonomy but also by the robustness, reactiveness and proactiveness of agents.

An agent can be relied upon to persist in achieving its goals, trying alternatives that are appropriate to the changing environment. This means that when an agent takes on a goal, the responsibility for achieving that goal rests with that agent. Continuous supervision and checking is not needed.

Agents tend to be used where the domain environment is challenging. Typically agent environments are *dynamic*, *unpredictable* and *unreliable* [Padgham and Winikoff, 2004]:

- *Dynamic*. These environments are dynamic in that they change rapidly. Thus the agent cannot assume that the environment will remain static while it is trying to achieve a goal.
- *Unpredictable.* It is not possible to predict the future states of the environment; often this is because it is not possible for an agent to have perfect and complete information about their environment.
- *Unreliable*. The actions that an agent can perform may fail for reasons that are beyond an agent's control.

In addition, agents in challenging environments might experience failure and thus recovery from failure must be done autonomously.

It has been argued [Jennings, 2001] that agents are 'well suited for developing complex distributed systems' since they provide more natural abstraction and decomposition of complex 'nearly-decomposable' systems.

MAS are able to cope with the intricacy (*e.g.*, uncertainty, approximate knowledge) related to the decision-making processes of complex real-world domains by integrating several agents that model real situations and work collaboratively for achieving the system's goals.

1.1.3 Rule-based Reasoning

Rule-based Reasoning is an AI technique which tries to emulate the human reasoning and problem solving capabilities. They model how a human expert analyzes a particular situation by applying rules to the facts in order to reach a conclusion.

A RBR system represents information and searches for patterns in that information. It contains a knowledge base and an inference engine which analyzes fact patterns and matches the applicable rules. Fact patterns are analyzed until either the goal succeeds or all of the rules are processed and the goal fails.

RBR has been widely and successfully applied to environmental management, supervision and control. Advantages or RBR are:

- *Declarative Programming.* Rules can express solutions to difficult problems and consequently have those solutions verified. Rules are much easier to read than code. Rule systems are capable of solving complex problems, providing an explanation of why decisions were made.

- *Logic and Data Separation.* The data is in the domain objects, the logic is in the rules. Logic can be much easier to maintain when changes occur, as the logic is expressed in rules.

- *Speed and Scalability.* The Rete and Leaps algorithms provide very efficient ways of matching rule patterns to the domain object data. These are especially efficient when you have datasets that change in small portions as the rule engine can remember past matches.

- *Centralization of Knowledge***.** An executable knowledge base is created and serves as a repository of knowledge.

- *Understandable Rules.* Rules should be written very close to natural language, so the logic can be understood easily by nontechnical domain experts.

1.1.4 The Belief-Desire-Intention model

Rational agents have an explicit representation of their environment and of the objectives they are trying to achieve. Rationality means that the agent will always perform the most promising actions (based on the knowledge about itself and the world) to achieve its objectives. As it usually does not know all of the effects of an action in advance, it has to deliberate about the available options.

Among the numerous deliberative agent architectures found nowadays the most widespread one is the Belief-Desire-Intention (BDI) proposed by Bratman as a model for describing rational agents [Bratman, 1987]. The concepts of the BDI-model were later adapted by Rao and Georgeff to a more formal model that is better suitable MAS in the software architectural sense.

The BDI architecture uses the concepts of belief, desire and intention as mental attitudes. Beliefs capture informational attitudes, desires motivational attitudes, and intentions deliberative attitudes of agents [Rao and Georgeff, 1995].

The advantage of using mental attitudes in the design and realization of agents and multi-agent systems is the natural (human-like) modeling and the high abstraction level, which simplifies the understanding of systems [McCarthy et al. 1979].

1.2 Objectives

The main goal of this thesis is the design and development of a MAS platform incorporating Rule-based Reasoning, as the main reasoning capability of agents to support decision-making

processes in real-world complex domains in a reliable way. The following objectives were foreseen:

- To design a domain-independent framework for supporting decision-making in complex real-world domains.

- To develop a software tool implementing the framework capable of:

    - Model the information about the complex domain.
    - Evaluating the consequences of critical processes in the decision-making.
    - Supervise the processes taking place at the domain.
    - To be able to simulate and predict the evolution of the system.

- The use of mature Artificial Intelligent techniques, such as Multi-Agent Systems and Rule- based Reasoning for the solution of complex problems.

- To extend a MAS tool capable of incorporating Rule-based Reasoning.

1.3 General overview

In Chapter 2 the State of the Art is presented. In Chapter 3 we describe our MAS framework. The validation of our work is presented in Chapter 4. Finally, in Chapter 5 we state the conclusions and contributions of this thesis and outline future work.

Chapter 2.

State of the Art

For the design and development of our *MAS-based framework* we revised the state of the art of MAS platforms [Mangina, 2002; Rendón et al., 2006], agent-oriented software methodologies [Dam and Winikoff, 2003; Dam, 2003; Sudeikat et al, 2004], and Decision Theory. The purpose of this revision was to select the most appropriate agent platform for our goals, and afterwards, to extend it with the new required functionalities. The most essential extension envisioned was the ability to incorporate knowledge to some agents using reasoning models, particularly Rule-based Reasoning.

After analyzing the MAS platforms we concluded that the most suitable platform for our goals was Jadex. An evaluation for the suitability of three agent-oriented methodologies, MaSE [DeLoach, 2001], Tropos [Myloppoulos et al, 2000] and Prometheus [Padgham and Winikoff, 2004], to the Jadex agent platform is presented in [Sudeikat et al, 2004]. The results illustrated that the three methodologies were capable of supporting the development of applications using Jadex. Moreover, Prometheus and Jadex proved to be the best match according to the criteria used. Thus, the authors concluded to propose the use of Prometheus for development with Jadex.

Why Jadex?

Jadex builds on experiences gained from leading existing BDI systems such as JACK [Winikoff 2005] and consequently improves previously not-addressed BDI weaknesses like the concurrent handling of inconsistent goals with built-in goal deliberation [Pokahr et al. 2005a].

This chapter presents the basic concepts of Multi-Agent systems, Decision Theory, and Rule Based Reasoning. In addition, the state of the art in Multi-Agent System platforms and agent-oriented methodologies is presented. Since we selected the agent platform Jadex and the Prometheus methodology among the myriad of options found in the literature we describe each of them in detail.

2.1 Agent Technology

2.1.1 Multi-Agent Systems

The most accepted and referenced definition of an agent in Artificial Intelligence is the one stated by [Wooldridge and Jennings, 1995] and then adapted in [Wooldridge, 2002]:

'An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives'.

Wooldridge's work distinguishes between and agent and an *intelligent* agent, which is further required to be autonomous, reactive, proactive and social [Wooldridge, 2002]:

- *Autonomous*: agents are independent and make their own decisions without direct intervention of other agents or humans and agents have control over their actions and their internal state.
- *Reactive:* agents need to be reactive, responding in a timely manner to changes in their environment.
- *Proactive:* an agent pursues goals over time and takes the initiative when it considers it appropriate.
- *Social:* agents very often need to interact with other agents to complete their tasks and help others to achieve their goals.

A key issue in agent architecture is balancing reactiveness and proactiveness [Padgham and Winikoff, 2004]. On the one hand, an agent should be reactive so its plans and actions should be influenced by environmental changes. On the other hand, an agent's plans and actions should be influenced by its goals. The challenge is to balance the two, often conflicting, influences: if the agent is too reactive, then it will be constantly adjusting its plans and not achieve its goals. However, if the agent is not sufficiently reactive, then it will waste time trying to follow plans that are no longer relevant or applicable.

Agents tend to be used where the domain environment is challenging; more specifically, typical agent environments are *dynamic, unpredictable* and *unreliable* [Padgham and Winikoff, 2004]:

- *Dynamic.* These environments are dynamic in that they change rapidly. Thus the agent cannot assume that the environment will remain static while it is trying to achieve a goal.
- *Unpredictable.* It is not possible to predict the future states of the environment; often this is because it is not possible for an agent to have perfect and complete information about their environment.
- *Unreliable*. The actions that an agent can perform may fail for reasons that are beyond an agent's control.

The term, multi-agent system, implies more than one agent interacting with each other within an underlying communication infrastructure and without a procedural control mechanism; and, the individual agents often are distributed and autonomous [Huhns and Stephens, 1999]. Multi-agent systems are based on the idea that a cooperative working environment can cope with problems which are hard to solve using the traditional centralized approach to computation. Intelligent agents are used to interact in a flexible and dynamic way to solve problems more efficiently.

Scores of problems in natural resources are inherently distributed both temporally and spatially. Many artificial intelligence-based methodologies, particularly those related to cooperative distributed problem solving and multiagent systems [Weiss, 1999] also are designed to address distributed problems.

[Anderson and Evans, 1994] discuss the application of intelligent agents as an approach to modeling in natural resource management, stressing the need for autonomy and the ability of an agent to interact spatially and temporally with surrounding entities. They also underscore the equal importance of providing a satisfactory representation of the spatial world in which the agents are embedded.

2.1.2 Agent Architectures

Agent architectures represent the move from theoretical specification to the software implementation [Mangina, 2002]:

Deliberative Architectures: Agents should maintain an explicit representation of their world, which can be modified by some form of symbolic reasoning. The Belief-Desire-Intention (BDI) model [Rao and Georgeff, 1995] is the most widespread model. Its central concepts are:

- *Beliefs:* Information about the environment.
- *Desires/Goals:* Objectives to be accomplished.
- *Intentions:* The currently chosen course of action.
- *Plans:* Means of achieving certain future world states.
- *Actions:* Ways the agent can operate on the environment.

Reactive Architectures: They aim to build autonomous mobile robots, which can adapt to changes in their environment and move in it, without any internal representation. The agents make their decisions at run time, usually based on a very limited amount of information and simple situation-action rule. Decisions are based directly on sensory input.

Hybrid Architectures: Many researchers suggested that a combination of the classical and alternative approaches would be more appropriate, as it would combine the advantages of both kinds and avoid the disadvantages. (eg. Procedural Reasoning System [Georgeff et al, 1987] and TOURINGMACHINES [Fergurson, 1992]).

Layered Architectures: These architectures represent the way different subsystems are arranged into a hierarchy of layers, which involve the different decompositions within the agents to complete their tasks.

2.1.3 Agent Communication

Agents have to communicate their knowledge for achieving their goals. Communication can be accomplished through the Agent Communication Languages (ACL), by using communication protocols including TCP/IP, SMTP and HTTP to exchange knowledge [Mangina, 2002].

The best known ACLs are the Knowledge Query Manipulation Language (KQML) [Finin et al, 1997], the Knowledge Interchange Format (KIF) [Genesereth, 1991] and the Foundation for Intelligent Physical Agents ACL (FIPA ACL).

- ***KQML*** It is both a message format and a message handling protocol to support run-time knowledge sharing among agents for cooperative problem solving. It employs a layered architecture of communication, where at the bottom the functionality for message transport or communication occurs and at the top the contents is specific by the application.

- ***KIF*** Apart from the communication language, agents need to have an understanding and parse the content of the messages they receive. This is facilitated by KIF, which provides a syntax for message content, which is essentially first order predicate calculus with declarative semantics.

- ***FIPA ACL*** aims to set general standards for agent interoperability. FIPA has defined an ACL (FIPA ACL), which includes basic communicative actions (inform, request, propose and accept) together with interaction protocols.

In distributed MAS the agents themselves may be transmitted across a computer network and executed remotely. Such agents are called mobile agents. In these architectures the agent software is transmitted to the host computer, the data are processed and the final result is communicated back.

2.1.4 Methodologies for MAS development

Software-engineering methodologies assume the existence of a set of concepts that it builds upon. For example, object-oriented notations such as UML (Unified Modeling Language) assume certain concepts such as object, class, inheritance, and so on. With agent-oriented methodologies, we also need an appropriate set of concepts, and it turns out that the set of concepts is different to the object-oriented set.

Since the selection of a right methodology is crucial for any software project [Sudeikat, 2004] we put especial emphasis on the choice of a suitable methodology for the agent platform selected. The agent-oriented methodologies found in the literature are:

- GAIA [Wooldridge, Jennings and Kinny 2000], studies the views definitions in a methodology and tries to integrate a software life cycle.

- MAS-COMMONKADS [Iglesias et al, 1998] due its origins (CommonKADS [Tansley and Hayball, 1993]), it is a methodology oriented to the development using practical experience in expert systems.

- MaSE [DeLoach, 2001], it has its own tool that supports the methodology. The primary focus of MaSE is to help a designer take an initial set of requirements and analyze, design, and implement a working multi-agent system.

- MESSAGE [Caire et al, 2002], defines the elements to take into account in the development of a MAS by means of the specification of meta-models. It proposes the adoption of a standard process, the RUP.

- PROMETHEUS [Padgham and Winikoff, 2004] is a mature and well documented methodology; it supports BDI-concepts and additionally provides a CASE-Tool, the Prometheus Design Tool (PDT), for drawing and using the notation. The Prometheus methodology consists in three phases that can be done simultaneously: system specification, architectural design and detailed design.

2.1.4.1 The Prometheus Methodology

Prometheus [Padgham and Winikoff, 2004] is a mature and well documented methodology; it supports BDI-concepts and additionally provides a CASE-Tool, the Prometheus Design Tool (PDT), for drawing and using the notation. The Prometheus methodology consists of three phases that can be done simultaneously: system specification, architectural design and detailed design. In Figure 1 an overview of Prometheus showing its three phases is depicted.

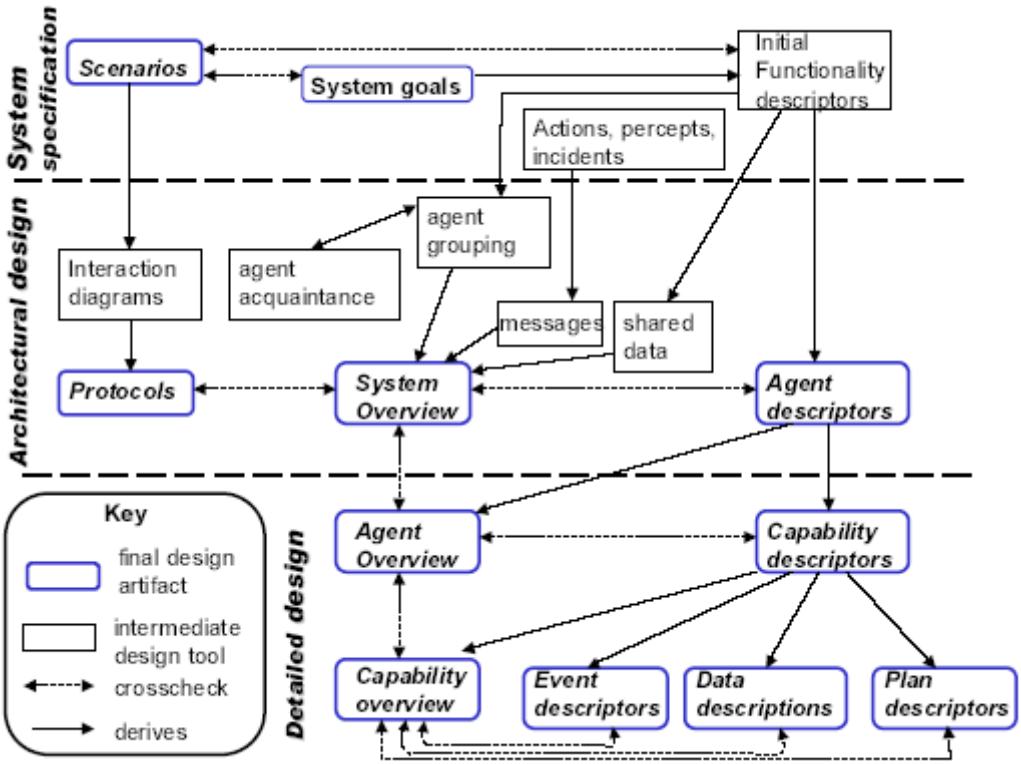

Figure 1. Prometheus overview (extracted from [Padgham and Winikoff, 2004])

- The first phase, *system specification* focuses on identifying the goals and basic functionalities of the system, along with inputs (percepts) and outputs (actions).
- The *architectural design* phase uses the outputs from the previous phase to determine which agent types the system will contain and how they will interact.
- The *detailed design* phase looks at the internals of each agent and how it will accomplish its tasks within the overall system.

System specification

This is the first phase; its main purpose is building the system's environment model, identifying the goals and functionalities of the system, and describing key use case scenarios.

Agents are situated in an environment that is changing and dynamic. Thus, building the environment model is an important step in this system specification stage. Modeling an environment involves: 1) identifying percepts which are incoming information from the environment and 2) determining actions which are the means by which an agent affects its environment. Percepts and actions are defined using descriptors and external resources such as data, information, etc. need to be identified.

Goals and functionalities of the system need to be captured at this stage. At the first step, system goals are identified mainly based upon the requirements specification. Goals are decomposed into subgoals if necessary. After that, system functionalities that achieve these goals are defined.

Architectural Design

There are three main activities involved in this intermediate phase: defining agent types, designing the overall system structure, and defining the interaction between agents. etermining which agents should exist in a system is an important step. This decision can be made by grouping the system functionalities previously defined in the system specification phase. Functionalities are grouped based upon two criteria: 1) functionalities that are related to each other are likely to be in the same group (cohesive criterion) and 2) if there are significant interactions between two functionalities, then there is a high chance that they should be grouped (coupling criterion). Prometheus also provides the data coupling diagram and agent acquaintance diagram as aids to the functionalities grouping process.

The system's structure needs to be captured in a system overview diagram, which is "arguably the single most important design artifact in Prometheus" [Padgham and Winikoff, 2004]. The system overview diagram is constructed based on the designers' understanding of the system up to this stage of the development process. It depicts the agent types and the communication links between them and the data used. Moreover, it shows the system's boundary and its environment in terms of actions, percepts and external data providing a general picture of how the whole system will function.

Detailed Design

The final stage of the Prometheus methodology is the detailed design. Here the internal structure and behavior of each agent are addressed. This stage emphasizes on defining capabilities, internal events, plans and detailed data structure for each agent type defined in the previous step.

Firstly, agent's capabilities are depicted with a capability descriptor which contains information such as which events are generated and which events are received. The capability descriptor also includes a description of the capability, details involving interactions with other capabilities and references to data read and written by the capability. Secondly, at a lower level of detail, there are individual plan, event, and data descriptors that provide the details so that they can be used in the implementation phase.
The detailed design phase also involves constructing *agent overview diagrams*. These diagrams are very similar to the system overview diagram in terms of style but give the top level view of each agent's internals rather than the system as a whole. Agent overview diagrams and capability descriptors, provide a high level view of the components within the agent internal architecture as well as their interactions. They show the top level capabilities of the agent, the flow of tasks between these capabilities and data internal to the agent.

Prometheus is supported by the Prometheus Design Tool (PDT), it provides forms to enter design entities. It performs cross checking to help ensure consistency and generates a design document along with overview diagrams.

2.1.5 Agent platforms for MAS development

A great number of agent platforms can be found in the literature. Roughly speaking, there are three classes of agent platforms:

1. Those that focus on internal agent reasoning and support plans, goals, and so on. Examples: PRS, UMPRS, JAM, JACK, DECAF, Zeus, AgentBuilder, JADEX.
2. Those that focus on inter-agent communications. These usually provide infrastructure for inter-agent communication, as well as facilities for locating agents (white pages)

and/or a description of the service that the agents provide (yellow pages). More recent platforms in this class tend to conform to the FIPA standards. Examples: JADE, Zeus, OAA.

3. Those that focus on mobile agents. Examples: Grasshopper, D'Agents, Aglets.

Although inter-agent communication is a specialty of the second class, all realistic platforms provide some support for agent communication. The first class is most useful in terms of providing support for implementing designs developed by following the Prometheus methodology [Padgham and Winikoff, 2004].

2.1.5.1 The Jadex platform

Jadex is a Java based framework that allows the creation of goal oriented agents and provides a set of development tools to simplify the creation and testing of agents.

The Jadex BDI reasoning engine allows the development of rational agents using mental notions. In contrast to all other available BDI engines, Jadex fully supports the two-step practical reasoning process: goal deliberation and means-end reasoning. This means that Jadex allows the construction of agents with explicit representation of mental attitudes (beliefs, goals and plans) and that automatically deliberate about their goals and subsequently pursue them by applying appropriate plans [Bellifemine et al., 2007].

The Jadex BDI reasoning engine enables the construction of complex-real-world applications by exploiting the ideas of intentional systems going back to Denmett and McCarthy.

2.1.5.2 Jadex Architecture

The abstract architecture of a Jadex agent is depicted in Figure 2. Practical reasoning is handled via *goal deliberation* and *means-end reasoning*. The goal deliberation is mainly state-based and has the purpose of selecting the current non-conflicting set of goals.

The means-end reasoning has incoming messages, internal events and new goals as input, and then it dispatches these events to plans selected from the plan library for further processing. Plans execution may access and modify the belief base, send messages to other agents, create new goals, and cause internal events.

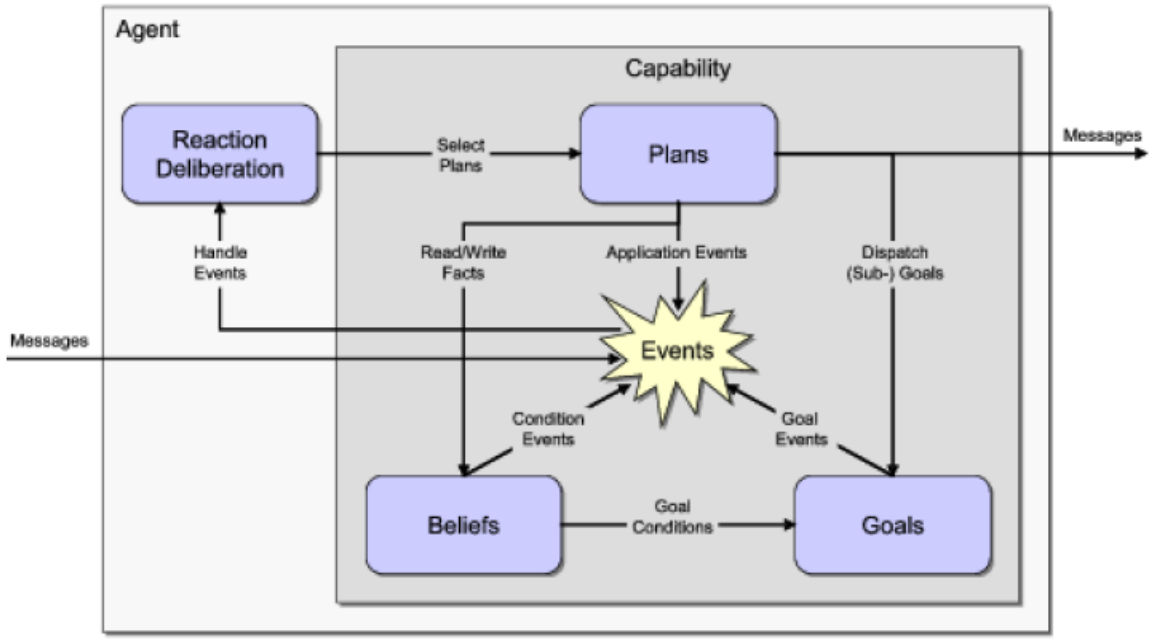

Figure 2. Jadex Abstract Architecture (extracted from [Jadex Tutorial])

2.1.5.3 Concepts for agent programming in Jadex

Jadex uses a hybrid approach for defining and programming agents. The structural part comprises the agent's static design composed of beliefs, goals, plans and the agent's initial state. All these aspects are specified in the Jadex XML language following an XML-Schema defining the BDI metamodel. The behavioural part of the BDI agent is encoded in Jadex plans using plain Java.

Beliefs

Beliefs represent the agent's knowledge about the world, itself and other agents. In Jadex, the belief representation is very simple, and currently does not support any inference mechanism. The beliefbase contains strings that represent an identifier for a specific belief. These identifiers are mapped to the beliefs values, called facts, which in turn can be arbitrary Java objects. Currently two classes of beliefs are supported: simple single-fact beliefs, and belief sets.

The belief base can be manipulated by setting, adding or removing facts using the belief names. Additionally, a more declarative way of accessing the belief base is provided by OQL-like queries. OQL (Object-Query-Language) is an extension of SQL (Structured-Query-Language) for object-oriented databases. Belief changes are automatically monitored by the Jadex engine as they may trigger a goal's creation or drop condition, or lead a plan to abort.

Goals

In Jadex, Goals play a key role; they are the driving force behind an agent's actions. Four types of goals are supported by the Jadex system: perform, achieve, query and maintain goals.

- A *perform goal* states that something should be done but may not necessarily lead to any specific result.

- The *achieve goal* describes an abstract target state to be reached, without specifying how to achieve it. Therefore, an agent can try out different alternatives to reach the goal.
- The *query goal* is used for information retrieval.
- The *maintain goal* specifies a state that should be kept or maintained under all circumstances. Whenever a specified situation is violated the agent will activate any applicable means to re-establish the desired world state.

With Jadex's *Easy Deliberation* strategy [Pokahr et al., 2005] goal cardinalities and inhibition links between goals can be modeled at design time, with the system ensuring that during run-time only valid goal subsets are active. If a goal set contains conflicting goals, the system will exploit the defined inhibition links to delay less important goals while executing the more important ones. Whenever goals are finished, the system considers the reactivation of currently inhibited goals.

Plans

Means-end reasoning is performed with the objective of determining suitable plans for pursuing goals or handling other kinds of events such as messages or belief changes. Procedural Reasoning Systems (PRS) such Jadex use the plan-library approach to represent the plans of an agent. A plan consists of two parts: the plan head and the plan body. The plan head contains information about the situations in which the plan will be used. Moreover, it contains the events and goals the plan can handle and conditions used to restrict the applicability. The plan body describes the actions that will be performed when the plan is executed.

The abstraction degree of a plan varies between very concrete and fully abstract. Concrete plans consist of directly executable actions, whereas fully abstract plans are specified in terms of subgoals only.

The plan head should be defined in the Agent Definition File (ADF) and the programming of the plan body in a pure Java class. This class extends the existing Jadex framework class, Plan.

Agent Definition

The complete definition of an agent is captured in a so called *agent definition file* (ADF). The ADF is an XML (Extensible Markup Language) file, which contains all relevant properties of an agent (e.g. the beliefs, goals and plans). In addition to the XML tags for the agent elements, expressions can be used in a Java-like syntax for specifying belief values and goal parameters. The ADF is kind of a class description for agents: From the ADF agents get instantiated like Objects get instantiated from their class.

2.2 Decision Theory

The basis for most models of management decision-making comes from Simon's model of problem solving [Simon, 1960]. This model depicts the problem-solving process as a flow of events that can proceed in either a linear or iterative fashion, allowing the decision maker to return to the previous steps for additional refinement. The phases of Simon's model are:

1. Intelligence: The process begins with this phase. During this phase the decision maker is looking for information or knowledge suggesting the presence of a problem or the need for a decision.

2. Design: Once the problem has been identified and defined, the decision maker analyzes different alternatives to solve the problem detected. This is a critical phase in the process. Each potential solution must be carefully analyzed and compared to all others with regard to a specific criteria deemed by the decision maker. (e.g. expected outcome, cost, probability of success).

3. Choice: During this phase, the decision maker selects one of the available solutions generated and analyzed in the Design phase.

Simon points out the importance of identifying the problem before trying to solve it. The decision itself is the culmination of the process. Regardless of the problem, the alternatives, the decision aids, or the consequences to follow, once a decision is made, things begin to happen. Decisions trigger action, movement, and changes.

2.3 Decision Support Systems

According to [Fox and Das, 2000], a decision support system is a computer system that assists decision-makers in choosing between alternative beliefs or actions by applying knowledge about the decision domain to arrive at recommendations for the various options. It incorporates an explicit decision procedure based on a set of theoretical principles that justify the "rationality" of this procedure.

A Decision Support System (DSS) is a system under the control of one or more decision makers that assists in the activity of decision-making by providing a set of tools intended to improve the effectiveness of the decision outcome. The decision maker should note that although the DDS is a valuable tool in the decision process is not a mechanism for the making of the decision itself. The components of a DSS are:

1. The data management system
2. The model management system
3. The knowledge engine
4. The user interface
5. The user

The concept of DSS was born in the early 1970s and is generally attributed to two articles written at that time: 1) "Models and Managers: The Concept of a Decision Calculus" [Little, 1970] and 2) "A Framework for Management Information Systems" [Gorry and Morton, 1989]. The first article described the concept of a decision calculus as a "model-based set of procedures for processing data and judgments to assist a manager in his decision-making". The author suggest that for such a system to succeed it must be simple, robust, easy to control, complete on issues of importance, adaptive to the needs of its user, and easy to communicate with.

The second article presents the concept of *decision support system*. The authors developed a two-dimensional framework for the provision of computer support to management activities. Both dimensions are assumed to be continuous rather than composed of discrete components. The vertical dimension is a classification of decision structure as originally proposed by Simon in [Simon, 1960].

2.3.1 Intelligent Decision Support Systems

An Intelligent Decision Support System could be defined as:

- R. Sojda (Sojda, 2002) defines the as systems using a combination of models, analytical techniques, and information retrieval to help develop and evaluate appropriate alternatives (Adelman 1992; Sprague and Carlson 1982); and such systems focus on strategic decisions and not operational ones. More specifically, decision support systems should contribute to reducing the uncertainty faced by managers when they need to make decisions regarding future options (Graham and Jones 1988). Distributed decision making suits problems where the complexity prevents an individual decision maker from conceptualizing, or otherwise dealing with the entire problem (Boland et al. 1992; Brehmer 1991).

- An intelligent information system that reduces the time in which decisions are made in an environmental domain, and improves the consistency and quality of those decisions (Haagsma and Johanns, 1994), (Cortés *et al.*, 2001).

Decisions are made when a deviation from an expected, desired state of a system is observed or predicted. This implies a problem awareness that in turn must be based on information, experience and knowledge about the process. Those systems are built by integrating several artificial intelligence methods, geographical information system components, mathematical or statistical techniques, and environmental/health ontologies, and some minor economical components (see Figure 3).

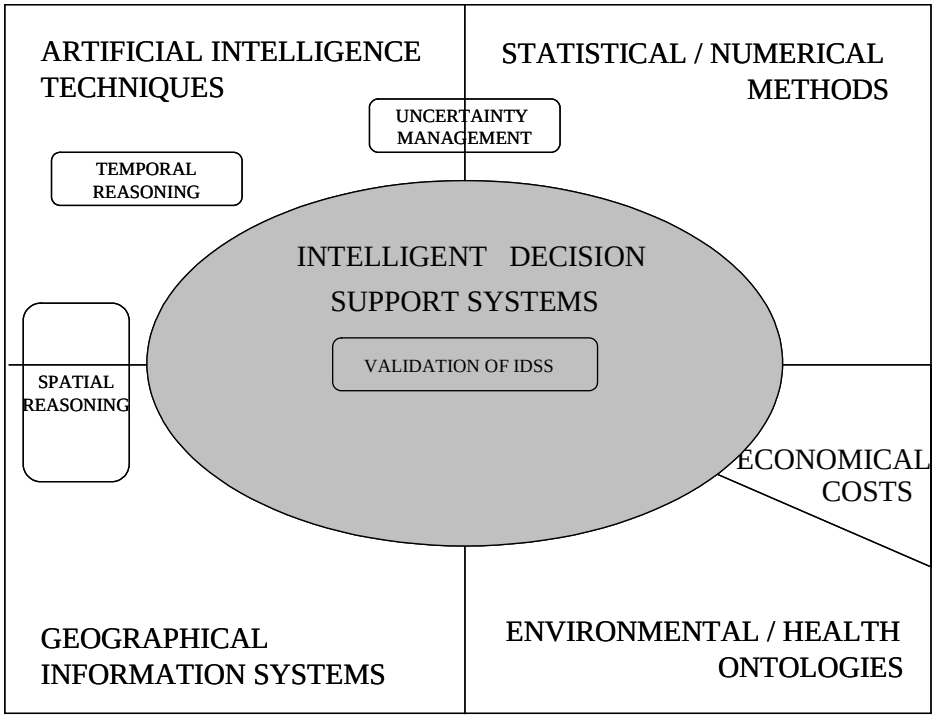

Figure 3. IDSS conceptual components

2.4 Knowledge representation and Reasoning

2.4.1Rule-based Reasoning

Production Rule System

A Production Rule System (PRS) focuses on knowledge representation to express propositional and first order logic in a concise, non ambiguous and declarative manner. A PRS uses an Inference Engine; this engine matches facts and data, against rules, to infer conclusions which result in actions (see Figure 4). A Production rule uses First Order Logic for knowledge representation and is structured in two parts:

If
    <conditions>
Then
    <actions>

Pattern matching is the process of matching the new or existing facts against rules. Inference Engines use different algorithms for Pattern matching like Linear, Rete, Treat and Leaps.

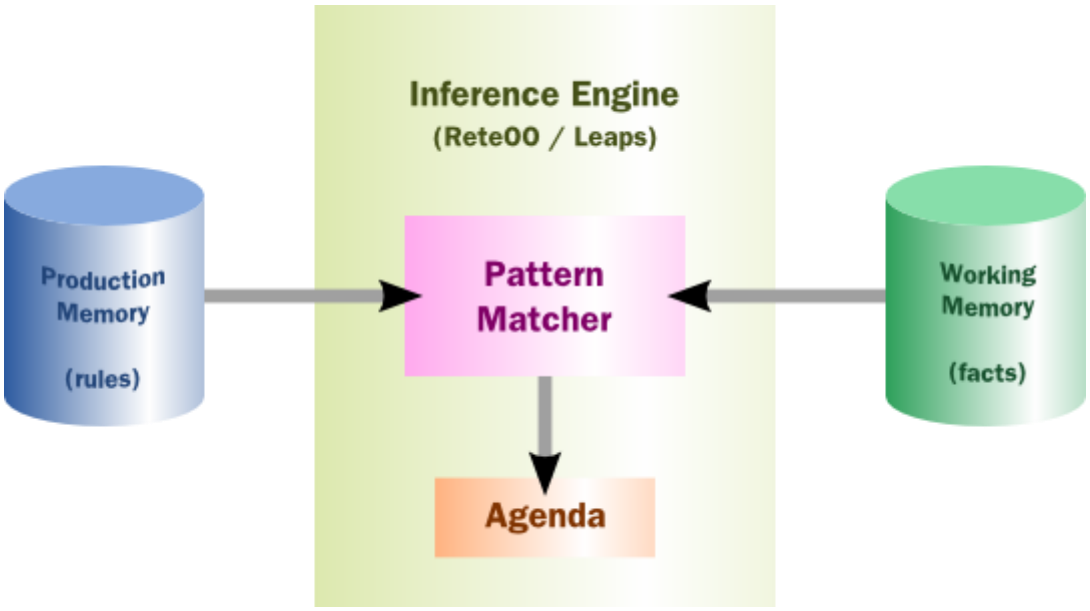

Figure 4. High-level View of a Rule Engine

Rete Algorithm

The Rete algorithm can handle pattern matching that involves several thousand working memory elements and rules without any severe resource requirements or unacceptable performance losses. This clever algorithm makes use of the fact that the contents of working memory do not change drastically after each rule application but rather exhibit only minor changes from the previous pass. Specifically, the Rete algorithm figures out which rules from the prior cycle did not fire, which rules from the prior cycle will not fire in the next cycle, and which rules from the prior cycle that did not fire will most likely fire in the next. Using this method, the algorithm avoids performing a pattern recognition cycle from scratch each time it

must cycle through the rules. It does this by maintaining an internal representation of the state of each rule in working memory and uses that representation as the basis for repeating the cycle during each subsequent pass until the pattern is matches.

Basic Rete network

Rules are stored in the Production Memory and facts in the Working Memory. Facts are asserted into the Working Memory where they can be modified or retracted. A system with a large number of rules and facts may result in many rules being true for the same fact assertion, these rules are said to be in conflict. The Agenda manages the execution order of these conflicting rules using a Conflict Resolution strategy.

There are two methods of execution for a Production Rule Systems: *Forward Chaining* and *Backward Chaining*. Forward Chaining is 'data-driven', it starts with a fact, it propagates and ends in a conclusion; facts are asserted into the working memory which results in one or more rules being true and scheduled for execution by the Agenda.

Backward Chaining is 'goal-driven'; it starts with a conclusion which the engine tries to satisfy. If it cannot then searches for conclusions that it can, sub goals, which will help to satisfy some part of the current goals – it continues this process until the initial conclusion is proven or there are no more sub goals.

2.4.1.1 The Drools Rule Engine

Drools is a Rule Engine implementation with a forward chaining inference engine based on the enhanced Rete algorithm and is tailored for the Java language.

Drools is split in two main parts: *Authoring* and *Runtime*. In the authoring process rules are stored in a DRL or XML file, which are fed into a parser. The parser checks for correctly formed grammar and produces an intermediate structure. This structure is then passed to a *Package Builder* which produces packages. See Figure 5.

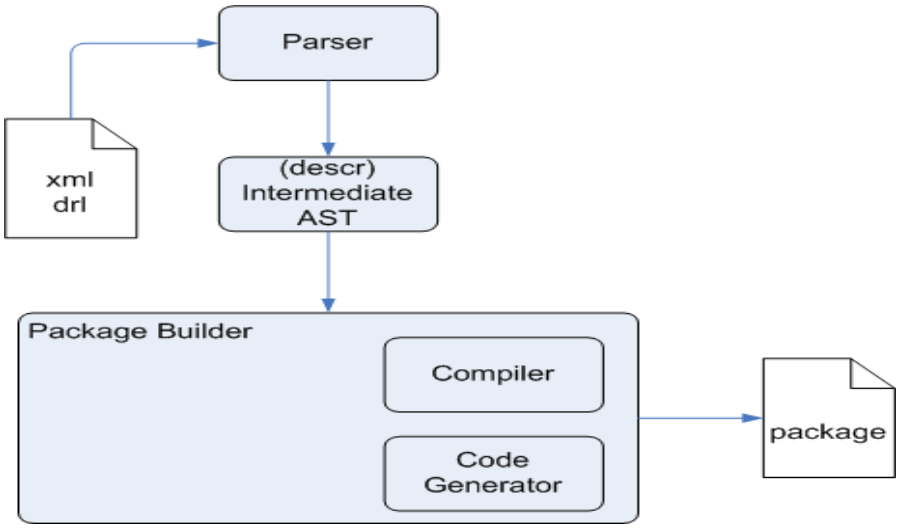

Figure 5. Authoring Components (extracted from [Drools Tutorial])

In the Runtime process we can find various components, see Figure 6. A *RuleBase* is a runtime component which consists of one or more Packages. They can be added and removed from the RuleBase at any time. A RuleBase can instantiate one or more Working Memories at any time;

The *Working Memory* consists of a number of sub components, including *Working Memory Event Support*, *Truth Maintenance System*, *Agenda* and *Agenda Event Support*. Object insertion may result in the creation of one or more Activations. The Agenda is responsible for scheduling the execution of these Activations.

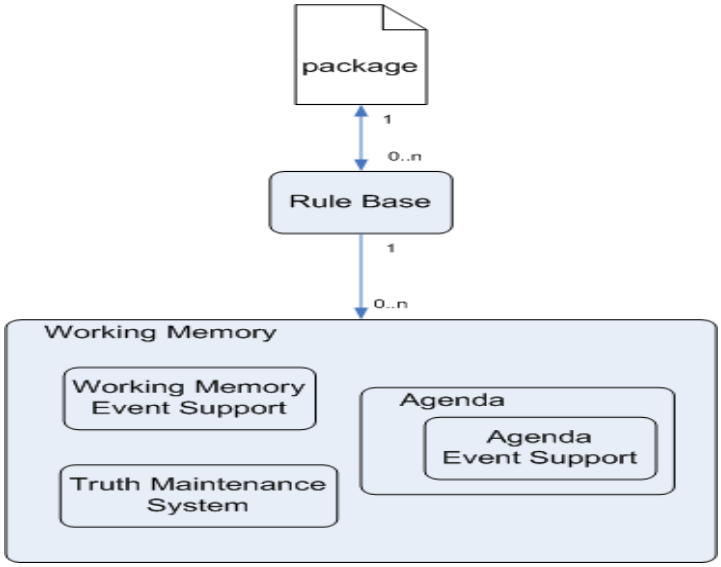

Figure 6. Runtime Components (extracted from [Drools Tutorial])

Engine Execution

The Agenda is a Rete feature. During a 'Working Memory Action' rules may become fully matched and eligible for execution; a single Working Memory Action can result in multiple eligible rules. When a rule is fully matched an Activation is created, referencing the Rule and the matched facts, and placed onto the Agenda. The Agenda controls the execution order of these Activations using a Conflict Resolution strategy.

The engine operates in a recursive two-phase mode (see Figure 7):

1. *Working Memory Actions*. This is where most of the work takes place - in either the Consequence or the main java application process. Once the Consequence has finished or the main Java application process calls the method `fireAllRules()` the engine switches to the Agenda Evaluation phase.

2. *Agenda Evaluation*. The Agenda attempts to select a rule to fire, if a rule is not found it exits, otherwise it attempts to fire the found rule, switching the phase back to Working Memory Actions and the process repeats again until the Agenda is empty.

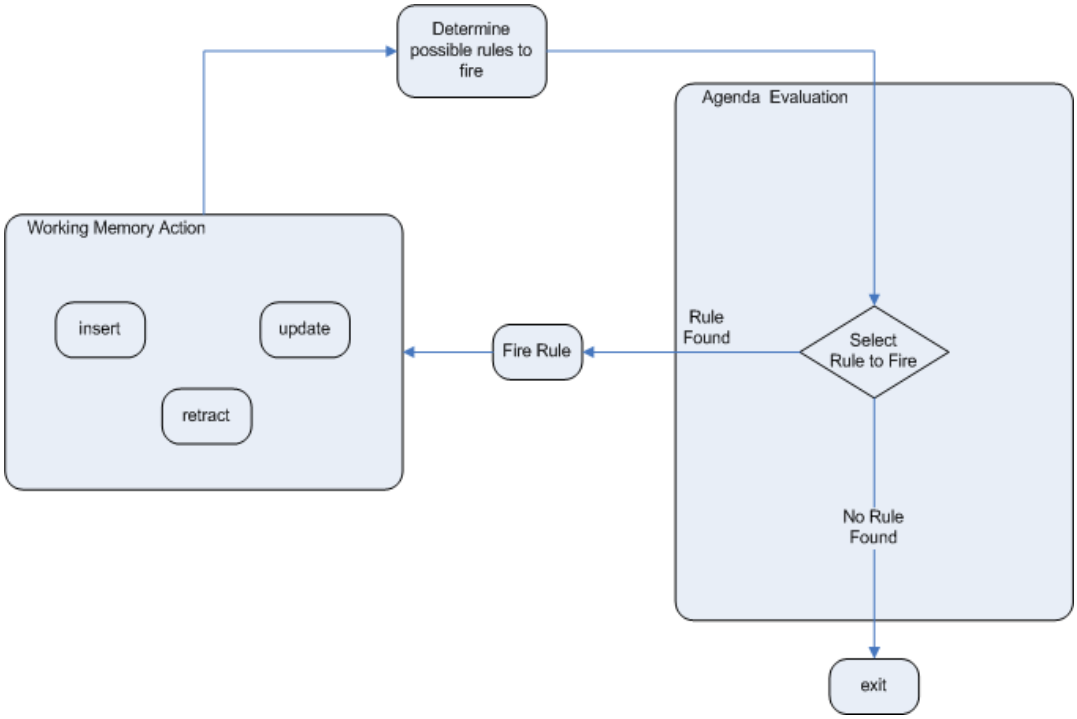

Figure 7. Two Phase Execution (extracted from [Drools Tutorial])

The recursive process finishes when the agenda is clear, then the control returns to the calling application. When Working Memory Actions are taking place, no rules are being fired.

Truth Maintenance

Given a set of facts the rule engine applies the appropriate rules and modifies the facts as necessary. If the newly-changed facts mean that other rules come into play, then they are applied. Eventually, all necessary rules are fired, and the facts stored in the working memory represent the truth, or at least the truth as understood by the rules.

Conflict Resolution

When there is one or more Activations on the Agenda they are said to be in conflict, and a conflict resolver strategy is used to determine the order of execution. The default conflict resolution strategies employed by Drools are: *Salience* and *LIFO* (Last In, First Out). With salience or priority, the user can specify that a certain rule has a higher priority than other rules. Therefore, the higher salience rule will always be preferred. LIFO priorities based on the assigned Working Memory Action counter value, multiple rules created from the same action have the same value - execution of these rules is considered arbitrary.

Chapter 3.

Design and Development of the Framework

The objective of this thesis was to develop a framework that supports the decision-making processes in complex real-world domains (*e.g.*, environmental, industrial or medical domains) using mature Artificial Intelligence techniques such Multi-Agent Systems and Rule Based Reasoning.

The framework considered the design and development of a software tool that could simulate different scenarios for evaluating the consequences of critical processes in decision-making.
In order to design and develop a prototype for our case study we selected the Prometheus methodology [Padgham and Winikoff, 2004] and the agent platform Jadex [Braubach et al, 2004] respectively.

Furthermore, by experimenting with possible "what if" scenarios, the end user can benefit from exploring the response surface and the stability of the solution from the safety of a simulated environment where no actual damage can be caused to people property or the environment as a result of incorrect decisions.

3.1 Multi-Agent System Framework

Our framework proposal draws information from the user and the complex domain (*eg.*, environmental or medical domains) about the domain-characteristics and processes, the framework architecture is depicted in Figure 8. In addition to acquiring domain knowledge it is able to organize and represent it. Our proposal for the framework consists of the following steps:

1. Represent the knowledge base in two steps: a) create the domain objects (facts) as POJOs (Plain Old Java Objects) complying with the *JavaBean* specification, and b) create a Drools Rule Language (DRL) or XML (Extensible Markup Language) file with the rules.

2. Create a reasoning agent in the Agent Definition File (ADF) that will be the bridge between the Multi-Agent System and the Drools engine.

3. Define a service description offering the reasoning service; store the *RuleBase* and *PackageBuilder* as beliefs and specify the interaction protocols for the agent communication in the ADF.

4. Create a Plan for the reasoning agent where you can retrieve the engine components from the belief base, then load the *Working Memory* with the facts received from the initiator agent. Fire the rules, wait for the conclusions and send the results back to the calling agent.

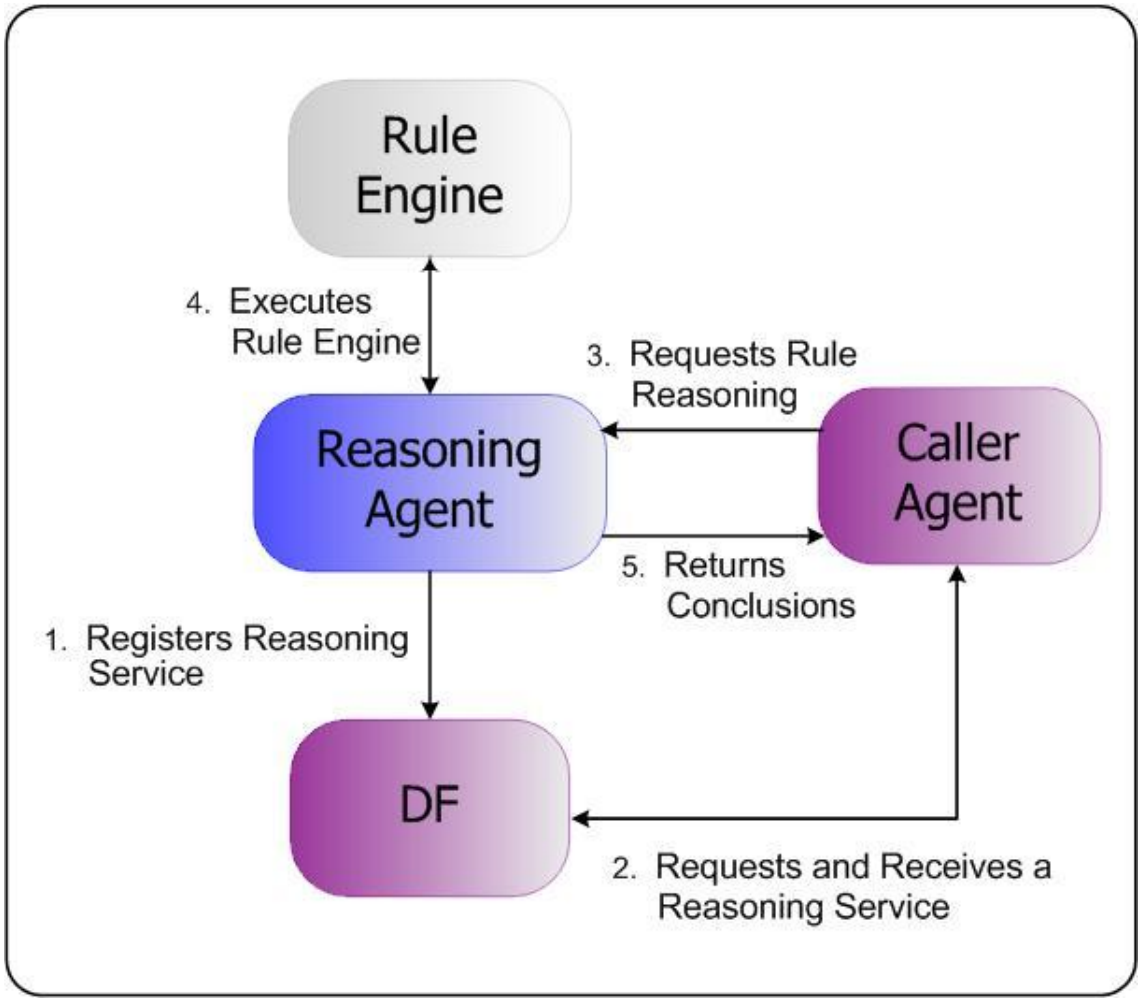

Figure 8. Framework Architecture

3.2 Integrating Drools within Jadex agents

Our goal was to extend the agent platform Jadex and provide it with an inference mechanism by means of Rule-based Reasoning. We integrated the Rete-based Drools Rule Engine to a reasoning agent that is able to perform deduction.

The aim was to take advantage of Jadex's full BDI functionality, including goal deliberation and means-end reasoning. In addition, we also wanted to provide a simultaneous inference mechanism by incorporating rule-based agents.

The process conducted for providing a Jadex agent with rule-based capabilities is explained next.

3.2.1 Knowledge representation

The knowledge is represented by means of rules and facts. Rules are specified in a file while facts in JavaBeans. The rule engine can create, delete and manipulate the objects (facts).

Facts Definition

Facts are our existing domain objects written in Java; they are represented as POJOs and are restricted to comply with the Java Bean specification. Facts are created using the class constructor. The class must contain getters and setter methods for each field so the engine can access and manipulate the object.

We assert our facts into Drools' Working Memory; the rule engine then uses these asserted facts to evaluate and map conditions.

JavaBean specification

JavaBeans are Plain Old Java Objects which are serializable, have a no-argument constructor and allow access to properties using getter and setter methods. The purpose of JavaBeans is to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects.

A *JavaBean* class must follow certain conventions about method naming, construction, and behavior. These conventions make it possible to have tools that can use, reuse, replace, and connect JavaBeans. The required conventions are:

- The class should be serializable. This allows applications and frameworks to reliably save, store, and restore the bean's state independently of the JVM (Java Virtual Machine) and platform.
- The class must have a public default constructor. This allows easy instantiation within editing and activation frameworks.
- The class properties must be accessible using get, set, and other methods following a standard naming convention. This allows easy automated inspection and updating of bean state within frameworks.

As these requirements are largely expressed as conventions some developers view *JavaBeans* as Plain Old Java Objects that follow specific naming conventions.

Rule Definition

Rules are represented in a DRL or XML file. A DRL file is simply a text file that can have multiple rules, queries and functions. Rules are written using First Order Logic; we can have conjunctive rules.

A rule is made of a conditional part and a consequence or action part. In Drools, the first part is called Left Hand Side (LHS), and the latter Right Hand Side (RHS); these two components form then "If-Then" rule defined as "When-Then":

```
rule "name"
      attributes
      when
            LHS <conditions>
      then
            RHS <actions>
end
```

A rule must have a unique name within its package. Attributes are optional; they influence the behavior of rules. The dialect attribute specifies the language to be used for code expressions, which is Java or Mvel.

The LHS consists of zero or more conditions. If the LHS is empty the condition evaluates to true. The RHS should contain a list of actions to be executed (e.g. insert, retract, update facts).

A rule condition is written to evaluate one or many of our fact objects. When the facts asserted into Working Memory fulfill all the rule conditions, the rule will activate, executing the consequence of that rule. The consequences defined within our rules can update the facts contained in our JavaBeans, stop the rule evaluation or execute a Java method.

For example we can see the rules defined in order to obtain the Fibonacci sequence. The Fibonacci Numbers are obtained by starting with 0 and 1, and then produce the next Fibonacci number by adding the two previous Fibonacci numbers.

The recurse rule is very simple, it matches each asserted Fibonacci object with a value of -1, it then creates and asserts a new Fibonacci object with a sequence of one less than the currently matched object. Each time a Fibonacci object is added, as long as one with a "sequence = 1" does not exist, the rule re-matches again and fires; causing the recursion. The 'not' conditional element is used to stop the rule matching once we have all 50 Fibonacci objects in memory. The rule also has a salience value, this is because we need to have all 50 Fibonacci objects asserted before the Bootstrap rule is executed.

Recurse and Bootstrap rules from the Fibonacci example:

```
rule Recurse
    salience 10
    when
        not ( Fibonacci ( sequence == 1 ) )
        f : Fibonacci ( value == -1 )
    then
        insert( new Fibonacci( f.sequence - 1 ) );
        System.out.println( "recurse for " + f.sequence );
end

rule Bootstrap
    when
        f : Fibonacci( sequence == 1 || == 2, value == -1 )
    then
        modify ( f ){ value = 1 };
        System.out.println( f.sequence + " == " + f.value );
end
```

3.2.2 Reasoning agent

The reasoning agent is the bridge between the agents and the Drools engine; it is depicted in Figure 9. The reasoning agent creates a service description in Jadex's Directory Facilitator (DF) offering its reasoning service. The *Rulebase* and the *PackageBuilder* are stored in the agent's belief base for the creation of a rule package and a rule base when the reasoning agent is born:

```
<belief name= "builder" class="PackageBuilder">
      <fact>new PackageBuilder()</fact>
</belief>

<belief name= "rb" class="RuleBase">
      <fact>RuleBaseFactory.newRuleBase()</fact>
</belief>
```

The agent should declare a Jadex plan where all the communication with the Drools engine will take place. This plan is executed when the agent receives a reasoning request via the FIPA Request protocol.

Inside the plan the agent obtains the domain facts from the `action` parameter of the request protocol. Additionally, it accesses its belief base and retrieves the rule base and the package builder. The rules are loaded from a source file (DRL or XML) in to the package builder that parses and compiles the rules. When the facts are asserted into the Working Memory the pattern matching process takes place; finally the rules are fired and the execution begins. Once the deduction finishes it returns the control to the application and sends the results (conclusions) back to the calling agent.

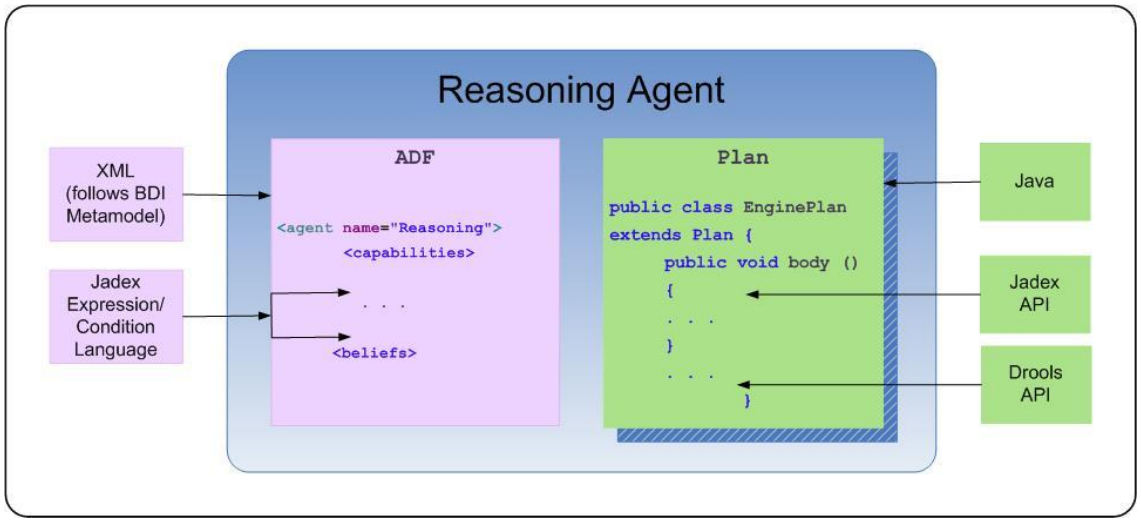

Figure 9. The Reasoning Agent

Rule Base

A rule base contains one or more packages of rules already validated and compiled. A rule base is instantitated using the *RuleBaseFactory*. Typically, a rule base would be generated and cached on first use, to save on the continually re-generation of the rule base. A rule base

instance is thread safe; you can have one instance shared across threads in the application. On the rule base you create a new rule session, either *stateful* or *stateless*.

We instantiate a *Rule Base* and create a *Package* using the beliefs declared in the Reasoning agent.

Sessions

There are two types of sessions: *Stateful* and *Stateless*. The *StatefulSession* extends the *WorkingMemory* class and allows iterative changes over time. A Stateless session does not use inference, it can be called like a function passing it some data and then receiving some results back.

Working Memory

The *Working Memory* provides access to the *Agend*a with methods for inserting, retracting and updating facts.

Inserting facts

The method `insert(object)` is used to assert a fact into the *Working Memory*. When a fact is inserted it is examined for matches against the rules. This means that all the work for deciding about firing or not firing a rule is done in this step. However no rule is executed until the `fireAllRules( )` method is called.

Retracting facts

Removing a fact from the *Working Memory* means that it will no longer be tracked and matched, thus any rule that is activated and dependent on that fact will be cancelled. Additionally, rules depending on the non-existence of that fact are activated. Retraction is done using a `FactHandle` that is returned by the insert call.

Updating facts

The `update( )` method is used to update facts and is only available within Java code. On the RHS of a rule the `modify` statement is supported, providing simplified calls to the object's setter methods.

Truth Maintenance

Given a set of facts the rule engine applies the appropriate rules and modifies the facts as necessary. If the newly-changed facts mean that other rules come into play, then they are applied. Eventually, all necessary rules are fired, and the facts stored in the working memory represent the truth, or at least the truth as understood by the rules.

Conflict Resolution

When there is one or more *Activations* on the *Agenda* they are said to be in conflict, and a conflict resolver strategy is used to determine the order of execution. The default conflict resolution strategies employed by Drools are: *Salience* and *LIFO* (Last In, First Out). With salience or priority, the user can specify that a certain rule has a higher priority than other rules. Therefore, the higher salience rule will always be preferred. LIFO priorities based on the assigned Working Memory Action counter value, multiple rules created from the same action have the same value - execution of these rules is considered arbitrary.

Drools Engine Execution

Drools is a forward-chaining rule engine; the execution is done in a recursive process:

- Uses Pattern matching to find activations.
- Enqueues activations in the Agenda.
- Selects an activation and executes its action.

3.2.3 Communication and Interaction between Agents

In Jadex, the communication between two or more agents is done through message events. Usually, these sequences of messages take place inside a conversation between some agents. The mechanism for handling conversations is based on the FIPA message structure, where three parameters are used for the conversation management: *conversation-id*, *in-reply-to*, *reply-with*. The parameter *conversation-id* has a unique value, and groups together several messages belonging to a single conversation. The *in-reply-to* parameter identifies a message as being an answer to a prior message with a corresponding *reply-with* parameter.

The process of managing these conversations can be tedious and error-prone as the programmer has to deal with the message interaction flow. Thus, as an alternative, Jadex offers the *Interaction Protocols* capability which implements some standardized FIPA interaction protocols. The interaction protocols capability allows the participating agents within a conversation to be programmed using abstract goals.

In our framework agents communicate using Interactions Protocols. So far we implemented the Request Protocol, however any protocol supported by Jadex can be used.

Coordination Protocols used

The Directory Facilitator (DF) Capability

Agents are allowed to register their services in the DF, and also to search for services offered by other agents.

The Interaction Protocols Capability

Interactions protocols are predefined patterns of message-based communication that are designed for interaction purposes. FIPA has specified standards for several domain-independent protocols. The interaction protocols capability offers implementations for most FIPA protocols:

- FIPA Request Interaction Protocol (RP)
- FIPA Contract Net Interaction Protocol (CNP)
- FIPA Iterated Contract Net Interaction Protocol (ICNP)
- FIPA English Auction Interaction Protocol (EA)
- FIPA Dutch Auction Interaction Protocol (DA)

FIPA Request Interaction Protocol (RP)

The Request Interaction Protocol (SC00026H) handles the interaction between an initiator and a participant agent. The initiator agent wants the participant agent to perform some action.

In our framework this translates to the caller and reasoning agents respectively. The initiator agent, asks the participant, to reason about something by sending a request message. The participant agent receives the message and can accept or refuse to perform the action requested. If it agrees then it uses the Drools rule engine to reason and when it finishes the participant

agent sends a message to the initiator. The message informs about the failure or success of the action (See Figure 10).

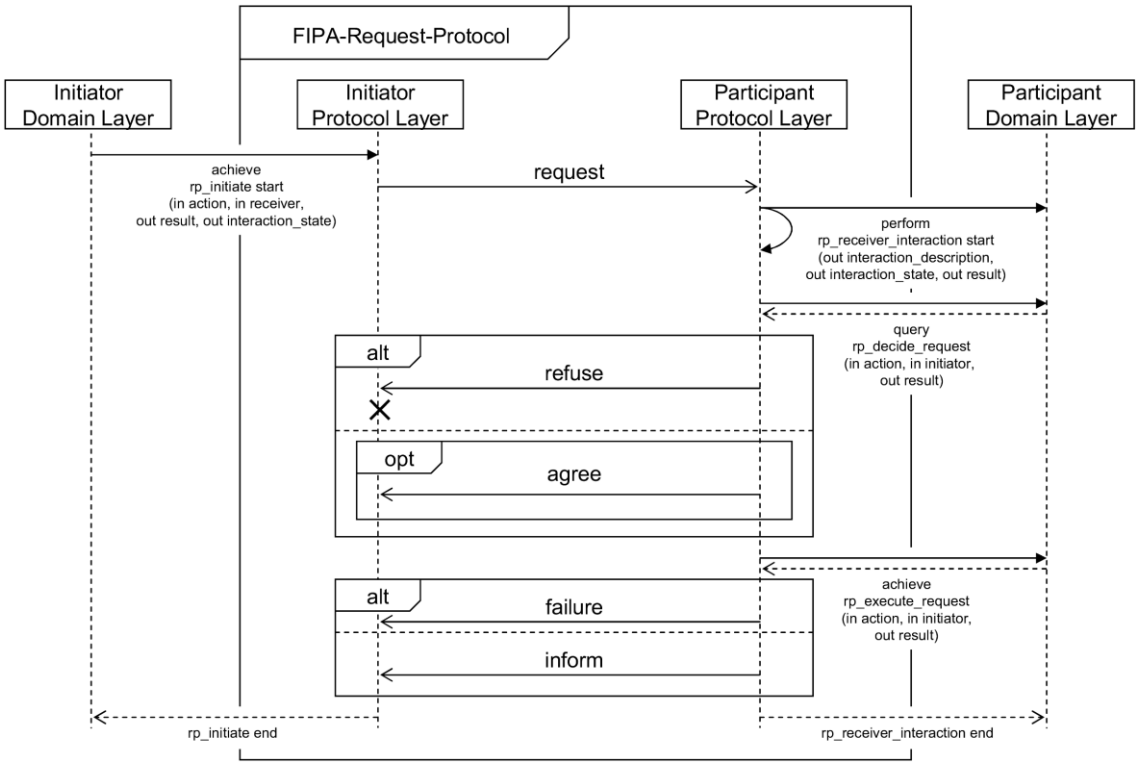

Figure 10. The FIPA Request Interaction Protocol

3.3 Modelling a complex real-world application

To illustrate the usage of our framework in complex real-world domains we modeled the river basin management from the environmental domain.

3.3.1 Waste Water Sanitation Systems

Wastewater management in river basins is becoming increasingly complex. Whilst there is an urge to reduce the ecological imbalances in fluvial ecosystems, more wastewater has to be treated because of both demographic and industrial growth. Furthermore, given the intrinsic multidimensionality of river basins, its management must take into account all the agents that affect and are affected by the wastewater.

3.3.2 River Basins Systems Management

River catchments are important social, economical and environmental units. They sustain ecosystems, which are the main source of water for households, agriculture and industry. Therefore the protection of all surface waters and groundwaters must be assured in their quality and quantity. The best way to fulfil these requirements is with a management system at catchment scale that integrates all the water systems involved (sewer system, Waste Water Treatment Plants and River) [Devesa 2006].

The management of river basins involve many interactions between physical, chemical and biological processes thus these systems become very intricate. Some of the problematic features found in the river basin domain are intrinsic instability, uncertainty and imprecision of data or approximate knowledge and vagueness, huge quantity of data, heterogeneity and different time scales to name a few [Poch et al., 2004].

3.3.3 Modelling

For instance, the following rule was defined in DRL for our case of study. The conditional part looks for an asserted object of type "`Pollutants`", it declares the `$cod` bound variable to use as a reference to the object in the consequences; also the `cod` field from our object is checked against a range of values. The consequence part prints a message and updates the field `performance` of the matched fact using the setter method `setPerformance` declared in the Pollutants' class.

```
rule "Range COD 6000 to 90000"
      when
            $cod: Pollutants(cod >= 6000, cod < 90000)
      then
            System.out.println("Performance 90%");
            $cod.setPerformance(0.90);
            update($cod);
end
```

Chapter 4.
Evaluation of the Framework

This section describes the results obtained in our work and illustrates the use of our framework. In order to show how an agent based framework is capable of supporting the decision making process in a complex real world domain we have modeled a river basin.

The prototype modeled features the main elements of the hydraulic infrastructure and aims to manage the environmental system as a single area, integrating the two sanitation systems (La Garriga and Granollers) with their respective sewage systems and WWTP's, as well as the Besòs river as the receptor for their waste water. Other elements are rain control stations, river water quality control stations, flow retention and storage tanks.

Later in this chapter we present the results of applying the framework described in the previous chapter to our case study. We observed that the results showed the following possible conclusions:

1) MAS are well suited to cope with complex domains.

2) Most agent-based platforms lack of an easy and explicit way to represent knowledge about the modeling domain and to reason about its behavior.

Overall, the evaluation shows that the prototype developed is able to support decision-making in complex real-world domains. Therefore, the framework prototype is useful, as it allows the user to draw conclusions from the different scenarios so that he can make better informed decisions.

4.1 Application in an environmental domain

The validation of our framework was done in the environmental domain, particularly in the management of river basins.

Wastewater treatment plants are large non-linear systems subject to large perturbations in flow and load, together with uncertainties concerning the composition of the incoming wastewater. Nevertheless these plants have to be operated continuously, meeting ever stricter regulations.

4.1.1 Case of Study: The Besòs River Basin

Our case of study is the Besòs river basin, located on the North East of the Mediterranean coast of Spain. The catchment area is one of the most populated catchments in Catalonia, having more than two million people connected. The scope of the study area is around the final reaches of the Congost River. The river sustains, in an area of 70 km^2, the discharges of four towns which are

connected to two WWTPs. The water system has three main elements: sewer system, WWTP and river depicted in Figure 11.

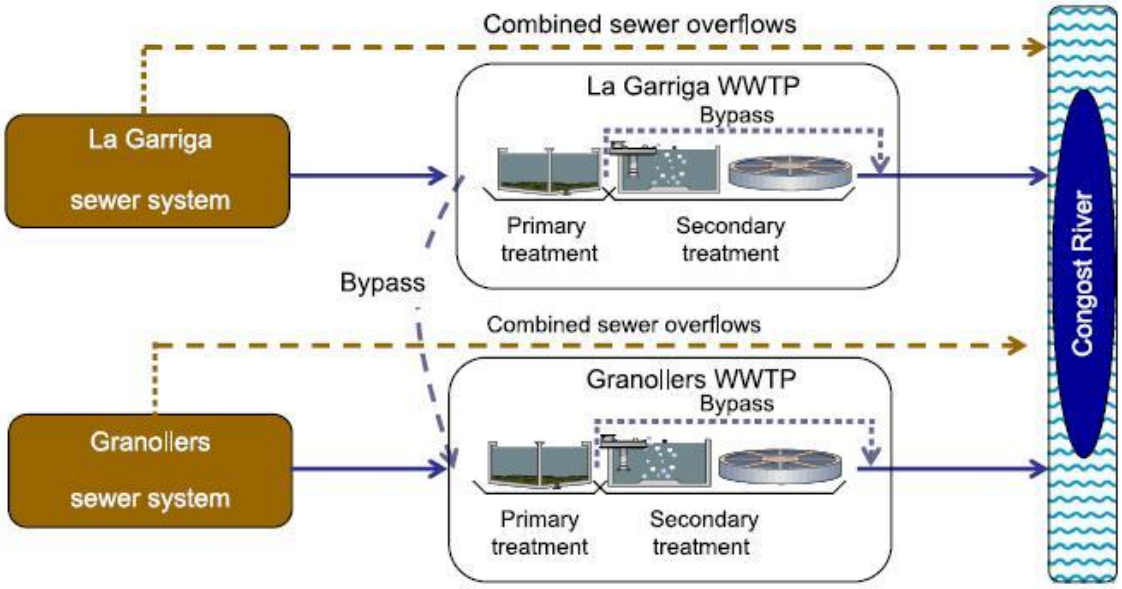

Figure 11. Elements of the case study

- ***Sewer system.*** There are two sewer systems, one that drains the area of the town La Garriga and another one that drains the area of Granollers and some small surrounding villages.

- ***WWTP.*** There are two WWTP, one for each sewer system. The two plants have a biological treatment. The average flows are 6000m^3/d for La Garriga (WWTP1) and 26000 m^3/d for Granollers (WWTP2).

- ***River.*** The studied reach of the Congost River has a length of 17 km. The Congost is a typical Mediterranean river with seasonal flow variations. Before the two WWTP, the average flow is about 0.5 m^3/s, but can easily reach a maximum punctual flow of 200 m^3/s.

Other considered elements are rain control stations, river water quality control stations, flow retention and storage tanks. Yet the most essential element is the sewer channel that joins the two WWTPs, allowing to by-pass the flow from the La Garriga-WWTP to the Granollers-WWTP [Devesa 2006].

4.1.2 Integral Management of Hydraulic Infrastructure Approach versus Traditional Approach

Traditionally hydraulic infrastructures for sanitation have been managed separately, taking into account only the characteristics of the water at the entry and exit points of each installation. The current tendency is to treat the hydrographic basin as a single area of operations. We are using this integrated approach because it presents more advantages to treat the environmental system as a whole instead of separate units.

Up to now, wastewater management has been supported by individual decision support systems. But the uncertainty or approximate knowledge involved in the processes along with distinct

spatial scales (i.e. local, regional, national, each associated with specific timescales) involved in wastewater management makes necessary to develop new architectures in which the decision making is being performed collaboratively among the different agents involved in all catchment's scales.

4.1.3 Benchmarking for control strategies for WWTPs

The benchmark is a simulation environment defining a plant layout, a simulation model, influent loads, test procedures and evaluation criteria. [http://www.benchmarkwwtp.org/]. For each of these items, compromises were pursued to combine plainness with realism and accepted standards.

We have three possible scenarios for a given WWTP defined in the benchmark:

- *Dry weather:* acts as the reference scenario; one can notice in the profiles the daily and weekly typical variations.
- *Rain weather:* (dry weather + long rain period), the scenario's data contains one week of dry weather and a long rain event during the second week.
- *Storm weather:* (dry weather + 2 storm events), the scenario's data contains one week of dry weather and two storms during the second week.

The scenario's data structure for the simulated plant has these components:

- *Time:* measured in days.
- *Flow:* measured in cubic meters per day (m^3/day).
- Concentrations of the following pollutants in milligram per liter (mg/l). The pollutants were defined by [Metcalf *et al*., 2003]:

    - ***COD:*** (Chemical Oxygen Demand). The COD test is used to measure the oxygen equivalent of the organic material in wastewater that can be oxidized chemically using dichromate in an acid solution, at high temperatures.
    - ***BOD5:*** (Biochemichal Oxygen Demand). It is a measure of the amount of oxygen required to stabilize a waste biologically.
    - ***TSSe:*** (Total Suspended Solids). Suspended solids can lead to the development of sludge deposits and anaerobic conditions when untreated wastewater is discharged to the aquatic environment.
    - ***TKN:*** (Total Kjeldahl Nitrogen). It is a measure of the organic and ammonia nitrogen.

In order to obtain the Final Concentration (C_f) for each pollutant the system uses the simple following equation:

C_f = (Initial Flow * Initial Concentration) + (WWTP Flow * WWTP Concentration)/Final Flow

$$C_f = (F_i * C_i) + (F_{wwtp} * C_{wwtp}) / F_f$$

where $F_f = F_i + F_{wwtp}$

Data Source

The following table shows an extract of the data characteristics obtained from the simulation of the WWTP's dry weather scenario; this data is later represented in our prototype as Java Beans. The influent data were initially proposed by Vanhooren and Nguyen. The time is given in days, the flow rate is given in m^3/day and the concentrations are given in g/m^3.

Scenario	Time	Flow	Concentration of pollutants in mg/l			
	T (days)	Q (m3/day)	COD (mg/l)	BOD5 (mg/l)	TSSe (mg/l)	TKN (mg/l)
"Dry"	0	21477	407,88755	205,983408	235,68975	54,44764
"Dry"	0,01041667	21474	405,87613	204,687295	235,65225	54,21328
"Dry"	0,02083333	19620	399,98873	204,394821	231,20175	54,45898
"Dry"	0,03125	19334	394,86518	202,138027	227,02875	54,77921
"Dry"	0,04166667	18978	391,76426	201,276881	222,89175	56,1501
"Dry"	0,05208333	18321	388,44071	200,799294	217,944	57,98396

The Java Beans are created by the WWTP agent using the Pollutants class constructor for each line read from the benchmark's file:

```
new Pollutants(scenario, time, flow, cod, bod5, tss, tkn, loadcod);

public Pollutants(String scenario,double time, double flow, double
cod, double bod5, double tss, double tkn, double loadcod) {
            super();
            this.scenario = scenario;
            this.time = time;
            this.flow = flow;
            this.cod = cod;
            this.bod5 = bod5;
            this.tss = tss;
            this.tkn = tkn;
            this.performance = 0;
            this.loadcod = loadcod;
      }
```

WWTP Performance function for COD

The WWTP performance is obtained using the rules defined by the expert. Assuming that the WWTP operates at an optimum level when it receives an average load of about 7000 Kg of COD per day. We can assume that a lower load of pollutants reduce the WWTP performance, perhaps slightly above the average load the performance is maintained or increased somewhat, but ultimately it also ends up reducing the performance due to overload.

River Basin Prototype

For the design and development of a prototype for our case study we selected the Prometheus methodology [Padgham and Winikoff, 2004] and the agent platform Jadex [Braubach et al, 2004] respectively. The prototype features the main elements of the hydraulic infrastructure and aims to manage the environmental system as a single area, integrating the two sanitation systems (La Garriga and Granollers) with their respective sewage systems and WWTP's, as well as the Besòs river as the receptor for their waste water. Other elements are rain control stations, river water quality control stations, flow retention and storage tanks. There also is a sewer channel that connects both WWTPs, allowing to by-pass the flow from the La Garriga-WWTP to the Granollers-WWTP.

Functionality of the Prototype

The aim of the MAS is to simulate various scenarios in order to draw conclusions and help in the decision-making for the river basin management. The goals to be fulfilled are:

- To manage critical episodes
- To minimize discharge of poorly treated wastewater
- To maximize the use of the installations treatment capacity
- To minimize the economical costs of new investments and daily management
- To maintain a minimum flow in the river guaranteeing an acceptable ecological state

In order to accomplish these objectives an intelligent agent will be provided with domain knowledge by means of RBR. Hence they can perform several tasks including:

- By-passing the water flow from La Garriga WWTP to Granollers WWTP.
- Storage tanks management
- Sewer system control
- Monitoring the river basin system

4.2 Design in the Prometheus methodology

In the *system specification* phase we modelled the environment by identifying the incoming information through percepts and determined the actions that the agents perform. Additionally the system goals and the basic roles of the system were identified. Figure 12 depicts the roles defined for our prototype. It shows the roles needed to fulfil the system goals. Roles are depicted by rectangles, goals by ovals and actions by an action icon.

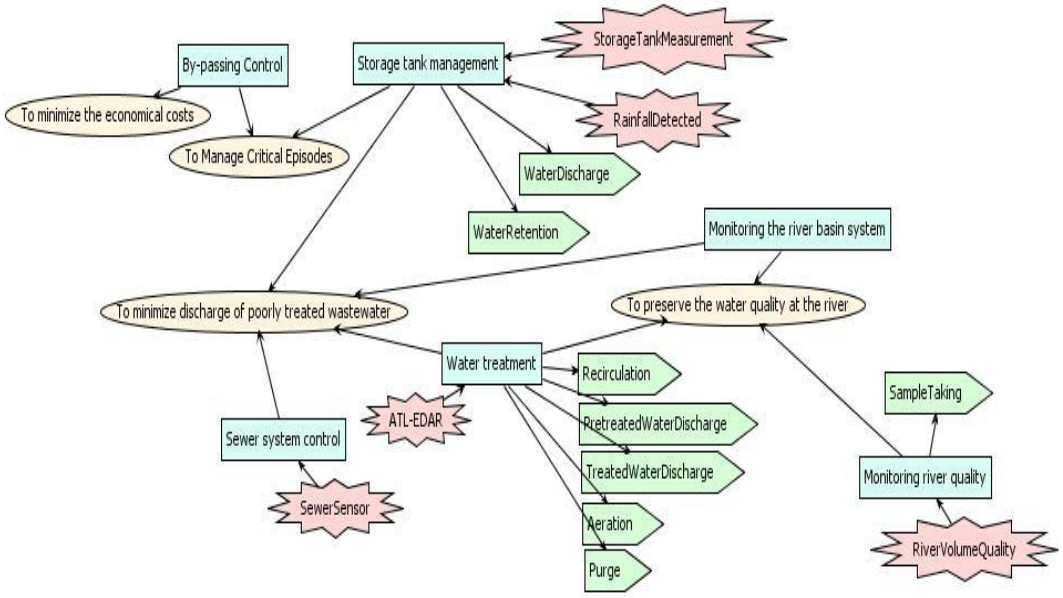

Figure 12. Roles Diagram built in Prometheus

An example of a Role descriptor is characterized in Table 2 for the Role Storage tank management.

Storage tank management –descriptor	
Name	Storage tank management
Description	1)It retains water during rain peak times and discharges it during low peaks. 2)Laminates waste waterflow. 3)Mitigates pollution episodes due to punctual discharges to the sewer
Percepts	RainfallDetected, StorageTankMeasurement
Actions	WaterDischarge, WaterRetention
Information used	SCADA
Goals	To manage Critical Episodes, To minimize discharge of poorly treated wastewater

Table 2. Role descriptor for Storage tank management.

4.3 Implementation in Jadex

For the implementation in Jadex we used a simplified version of our case study. Currently we have the Manager, WWTP, River, Environment, User, and, the Drools agents implemented.

Description of the Implemented Agents

Manager agent

The manager agent starts the application with all the system agents.

River Agent

This agent central role is data gathering (meteorological, physical, kinetic, water quality), for monitoring the state of the river; it provides the initial data for the water flow and the pollutant concentration according to the scenario selected.

Currently the River agent has in its belief base the initial values from the first control point in the river:

Scenario	Flow (m^3 s^{-1})	Description
Dry	0.01	Minimum flow
Rain	0.48	Rainfall
Storm	4.31	Heavy rain

It saves in Tuples the values for the water flow and the concentration of various pollutants. These initial values can be retrieved using the OQL queries defined in the agent:

```
<expression name="query_qi">
      select one $pair.get(1)
      from Tuple $pair in $beliefbase.qi
      where $pair.get(0).equals($escenario)
      <parameter name="$escenario" class="String"/>
</expression>

<expression name="query_ci">
      select one $pair.get(1)
      from Tuple $pair in $beliefbase.ci
      where $pair.get(0).equals($pollutant)
      <parameter name="$pollutant" class="String"/>
</expression>
```

The query selects the first matching entry, whereby the parameter `$escenario` or `$pollutant` is compared to the first element of the corresponding belief set Tuple.

Additionally, the River agent implements the RP Protocol; it acts as the participant side in the interaction. It returns the query results in the result parameter of the `"rp_execute_request"` goal.

WWTP agent

The WWTP agent provides the data characteristics from the benchmark for the sewage treatment plant. It reads the data from a CSV file and loads it as facts in a beliefset. This belief set is later accessed by the Drools agent in order to create its knowledge base.

This agent calculates the load for each pollutant according to this equation:
Load = (flow * concentration) * (1 – WWTP's performance percentage).

In addition, it provides the general characteristics for the influent and effluent water flow (as beliefs).

Drools agent

This is the reasoning agent; it is the bridge between the agents and the Drools engine. The Drools agent creates a service description "`drools engine`" and publishes it in Jadex's DF offering its reasoning service. When the Drools agent receives a request for its services it executes the Drools rule engine to deduct and returns the conclusions to the caller agent.

The Rule base and the Package builder are stored in the agent's belief base. When the Drools agent is born it creates a Rule Base and rule package respectively.

```
<belief name= "builder" class="PackageBuilder">
      <fact>new PackageBuilder()</fact>
</belief>

<belief name= "rb" class="RuleBase">
      <fact>RuleBaseFactory.newRuleBase()</fact>
</belief>
```

The Drools agent specifies a Jadex plan where all the communication with the Drools engine will take place. This plan is executed when the agent receives a reasoning request via the FIPA Request protocol.

Inside the plan the agent obtains the domain facts from the `action` parameter of the request protocol. Additionally, it accesses its belief base and retrieves the rule base and the package builder. The rules are loaded from a source file (DRL or XML) in to the package builder that parses and compiles the rules. When the facts are asserted into the Working Memory the pattern matching process takes place; finally the rules are fired and the execution begins. Once the deduction finishes it returns the control to the application and sends the results (conclusions) back to the calling agent.

User agent

This agent initiates the FIPA Request Protocol. The User agent requests the initial values for the flow and the pollutant concentration to the River agent (via the RP Protocol). It also searches for the reasoning service in the DF, and sends a request (via the RP Protocol) to the Drools agent.

When the Drools agent returns the conclusions, the user agent calculates the final concentration for the pollutant and generates and stores the data series for constructing the output charts in the Environment helper class.

Environment agent

This agent creates the Graphical User Interface (GUI). The interface has two main components: the "Wwtp Input" and "Simulation" tabs.

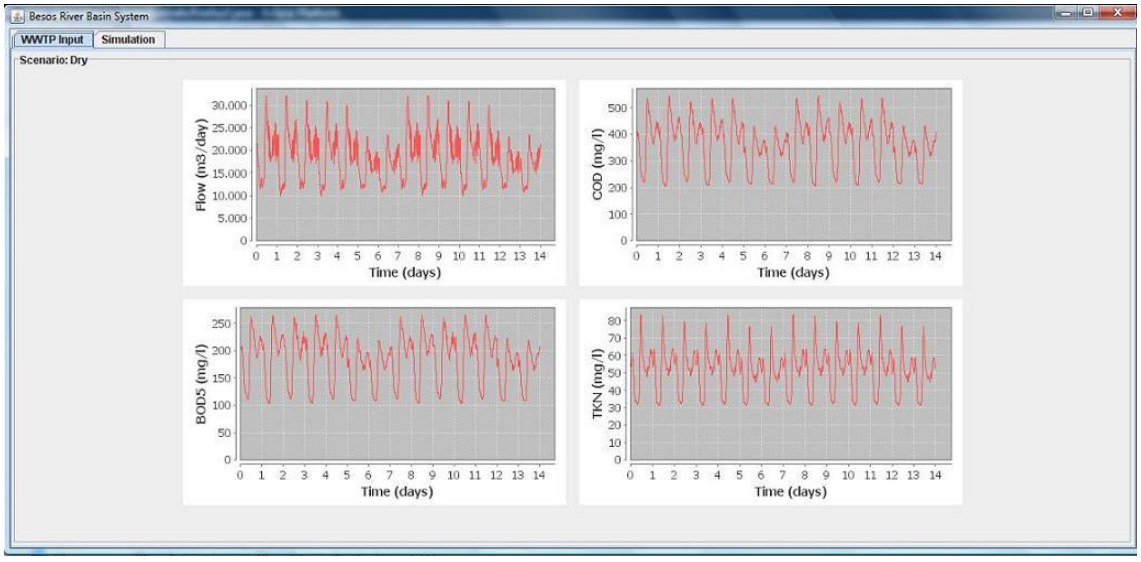

Figure 13. Prototype Screenshot showing the WWTP Input view.

The "Wwtp Input" tab shows the influent data of the WWTP taken from the benchmark (see Figure 13). The data is depicted by charts; each chart corresponds to the flow and pollutants at the entry point of the WWTP.

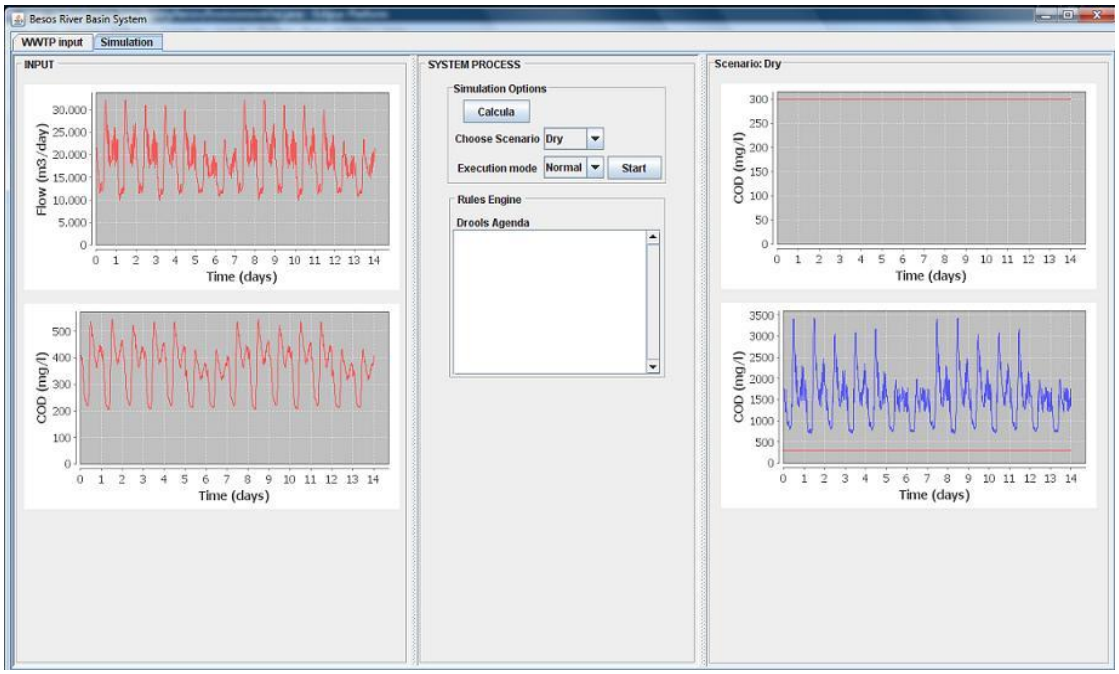

Figure 14. Prototype Screenshot showing the simulation view.

In Figure 14 we can notice that the Simulation view is divided in three sections: input, system process and output.

In the input section we can find two charts representing the flow and the pollutant selected for the WWTP (see Figures 15 and 16). The system process depicts the simulation options and the Drools execution. When the user selects a scenario (dry, rain or storm) the input information is updated according to the scenario selected. In addition, the user can choose between normal,

slow or step for the execution mode. Should the user click the start button the simulation is triggered.

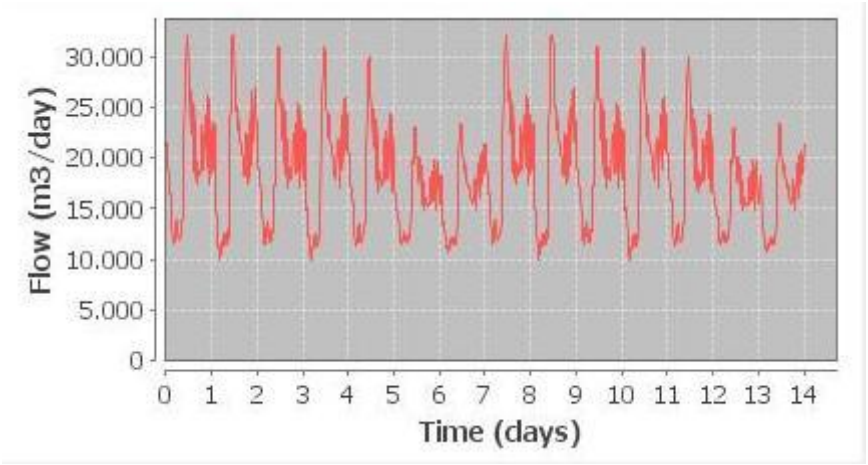

Figure 15. Flow at the entry of the WWTP (Dry weather)

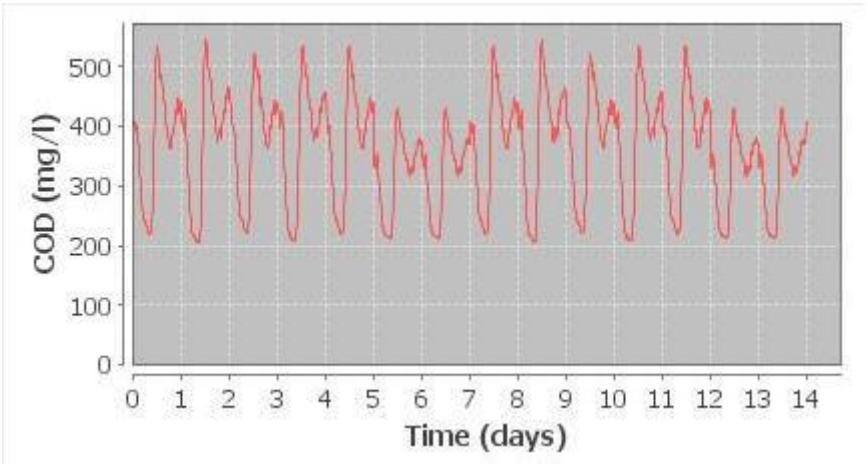

Figure 16. COD concentration at the entry of the WWTP (Dry weather)

The output view shows the simulation results for a given pollutant. It features two charts: the first one reflects the plant output (after applying the equation and performance function), and the second one depicts the river state before and after the wastewater treatment process. (See Figures 17 and 18).

Agent Communication and interaction in our Prototype

1. The manager agent starts the WWTP, Drools, River, Environment and User agents.
2. The Drools reasoning agent publishes its service in the DF, the GUI shows the system's views and options, and the rest of the agents initialize their belief bases.
3. The human user selects a scenario, an execution mode and clicks the start button; this broadcasts an internal message that triggers the simulation.
4. First the User agent requests the initial values for the flow and the pollutant concentration to the River agent (via the RP Protocol).
5. The River agent uses an expression to query its belief base and sends the result back to the User agent.

6. Then the User agent searches for services in the DF, finds the Drools service and sends a request (via the RP Protocol) to the Drools agent.
7. The receiver side of the RP Protocol, the Drools agent, receives and accepts the request.
8. The Drools agent creates the rule and knowledge bases, and then it uses the Drools' rule engine to perform the deduction. When it finishes it sends the result back to the User agent.
9. The user agent calculates the final concentration for the pollutant and generates and stores the data series for constructing the output charts in the Environment helper class.
10. The GUI shows the results in the output section.

4.4 Experimental Evaluation

In order to help evaluate and fully understand the meaning of data from the simulated scenarios we sought expert opinion.

On the basis of the results produced the expert was able to interpret the data and perform an analysis of the results from each simulated scenario. This analysis showed that relevant and useful knowledge can be obtained about the river water quality and that the knowledge can be used to support the final user (e.g. WWTP manager) in his/her decision-making.

The simulation results of the different scenarios were analyzed by the expert. The dry weather and stormy weather scenarios analysis are presented in this section.

Dry Weather Scenario Analysis

The dry weather scenario acts as the reference scenario; we can note the daily and weekly typical variations in the profiles.

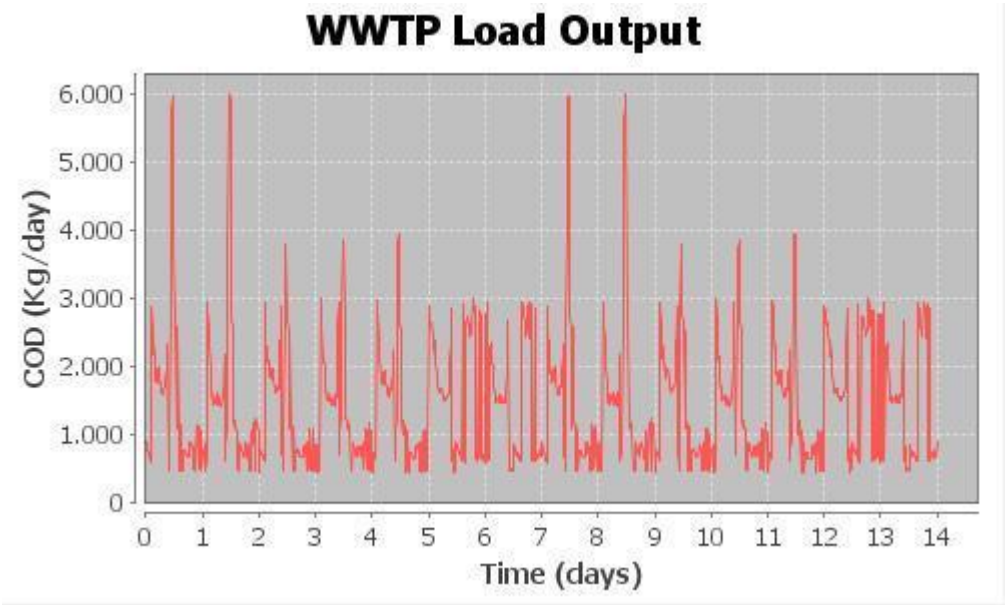

Figure 17. WWTP effluent for the Dry weather scenario.

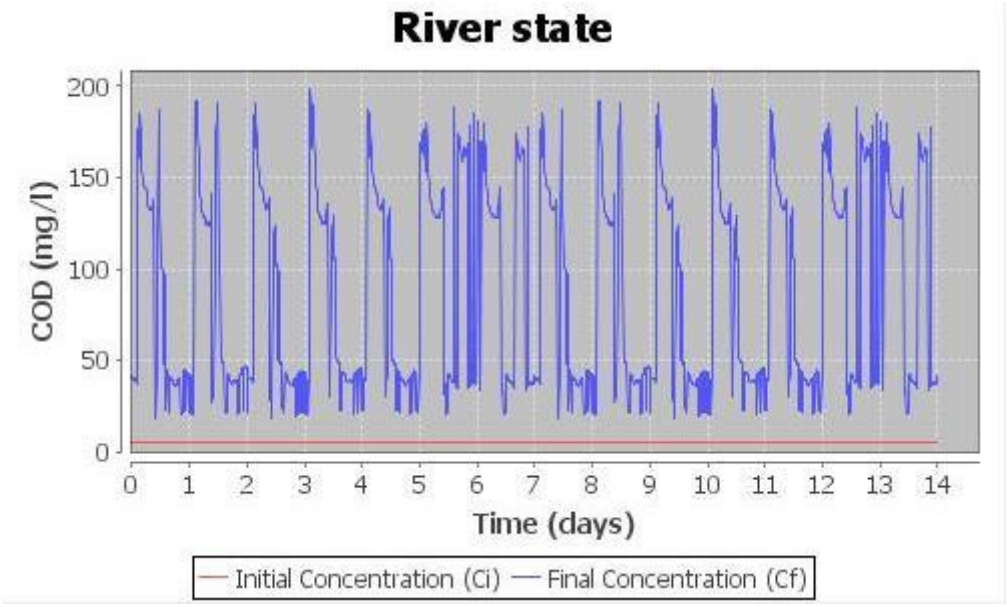

Figure 18. COD concentration in the river for the Dry weather scenario.

As regards the output charts shown in Figure 17 and 18, the expert concluded that each peak in which the WWTP has failed to eliminate the COD pollutant (See Figure 17) coincides with a peak on the chart of the river's final state. This shows the great influence that plant spills have on the river. This influence is due to several reasons:

- The river has very little dilution capacity because its flow is very low due to dry weather.
- Almost all the river flow after the WWTP spill comes from the spill flow itself. This is a typical behavior of Mediterranean rivers in dry weather where there is very little rain or, no rain at all.

An environmentalist's analysis could perhaps start by investigating whether or not the river state meets the legislative requirements, and what these values mean for the river life.

Storm Weather Scenario Analysis

The input data file contains one week of dry weather followed by two storms during the second week. Figures 19 and 20 show the WWTP effluent and the COD concentration in the river before and after the WWTP spill for the Storm weather scenario.

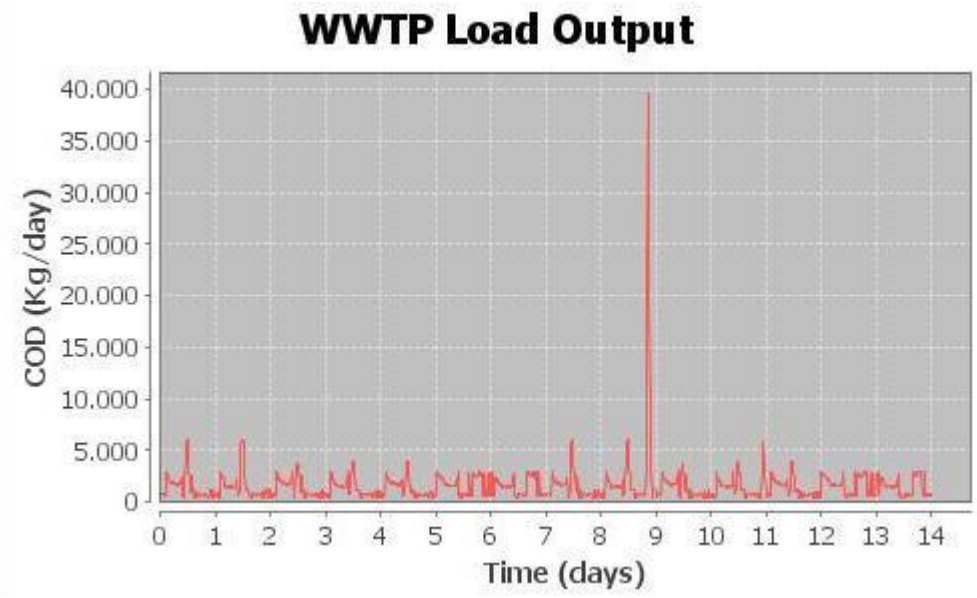

Figure 19. WWTP effluent for the Storm weather scenario.

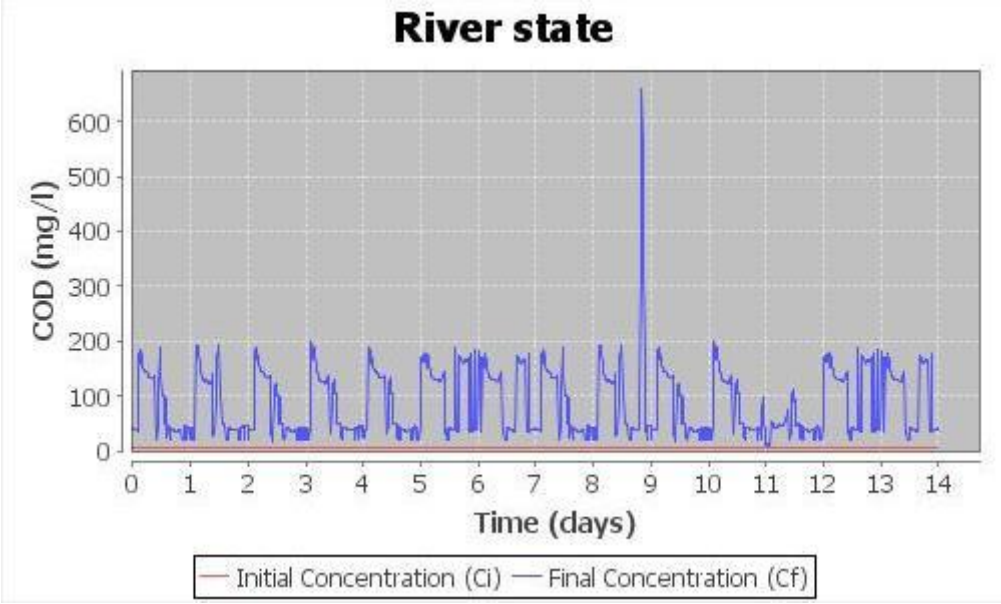

Figure 20. COD concentration in the river for the Storm weather scenario.

The basic hypothesis is that pollutants accumulate during dry weather periods before they are transferred into and along the system. The accumulation process occurs both on catchment surfaces and in sewer pipes. Pollutants accumulated on catchment surfaces are washed off during storm events.

Following a storm event large amounts of pollutants that were previously dormant in sediment deposits spread around the drainage system (sewers, streets etc) are carried to the WWTP all at the same time. If the WWTP lacks the capacity to process such large volumes of water then it has to be bypassed but, nevertheless, a very high concentration of pollutants will still enter the WWTP.

This surge of pollutants affects the performance of the WWTP and as such we can note on the graph of Figure 19 that the exit flows register a “high concentration peak” on day nine. On the

other hand, a prolonged period of rain also increases the river's capacity to dilute pollutants and so this explains why there is just one peak after which the load of pollutants diminishes very fast.

We should also note that the same effect is seen in the final concentration of pollutants in the river, i.e. an extraordinary peak of pollution concentration. As a result of this peak, the river has very little dilution capacity which, indeed, is the same as would be the case during a dry period. What we then see is that the water discharged from the WWTP can make the final concentration of COD in the river increase substantially, which in turn can lead to the levels of oxygen in the river diminishing considerably, a phenomenon known as "oxygen depletion".

Chapter 5.
Conclusions and Future Work

In this thesis we have shown how an agent-based framework capable of supporting the decision-making process was developed for use in complex real-world domains.

We believe that the conclusion above supports the objectives as stated in the Introduction:

- To build a domain-independent framework for supporting decision-making in complex real-world domains.
- To extend a MAS tool capable of incorporating Rule-based Reasoning.

In order to meet these objectives we conceptualised and explained our framework in Chapter 3. Later, in Chapter 4, we illustrated the application of our framework and described the design and development of our prototype as well as the validation and preliminary results.

In summary, our framework draws information from the user and the complex domain (*e.*g., environmental, industrial or medical) about the domain-characteristics and processes. In addition to acquiring domain knowledge the framework is able to organize and represent the information.

The goal of our research undertaken in this thesis was to extend the agent platform Jadex and provide it with an inference mechanism by means of Rule-based Reasoning. Additionally, we aimed to exploit Jadex's full BDI functionality, including goal deliberation and means-end reasoning. Finally, we also wanted to provide a simultaneous inference mechanism by incorporating a rule-based agent.

In order to accomplish the goals outlined above, we have extended the Jadex platform by integrating the Rete-based Drools Rule Engine to a reasoning agent that is able to perform deduction. Finally, we have also built a prototype of our case of study using our extension.

As a means of illustrating the application of our framework in a complex real-world domain we selected the environmental field and thus modeled a prototype of our case of study, the Bèsos river basin.

In order to test the validity of our approach we tested the prototype with data from the benchmark for control strategies for WWTPs. We simulated a set of scenarios and the results obtained were then analyzed, focusing on the river water quality, with the main objective being to gain relevant knowledge to deal with the scenarios tested.

Expert analysis of the results of the simulated scenarios showed that relevant and useful knowledge can be obtained about the case study. This knowledge will support the final user (e.g. WWTP manager) in his/her decision-making. Furthermore, expert evaluation found that the prototype is useful for making informed decisions regarding the river basin management.

In final conclusion, we have shown how agents are able to cope with the complexity of real-world domains and, in particular, how MAS can successfully incorporate RBR. We have proved the usefulness of our approach by applying our framework to facilitate good management of the Bèsos river basin. In conclusion, we believe that this work represents an interesting step for future research in this field.

5.1 Future Work

There are some lines of research and development to be done in the future. For instance the following items are envisioned:

- To do more experimentation in the same domain and in other domains (e.g. medical, industrial) to ensure a general framework.
- To add backward reasoning capability to the rule engine of our framework to enhance its reasoning abilities.
- To integrate new reasoning skills to the MAS, such as Case-based Reasoning (CBR) to get a more reliable and useful MAS tool for modelling real-world complex domains.

Appendix A

Publications

- Rendón-Sallard T. and Sànchez-Marrè M., Multi-Agent prototype for simulating scenarios for decision-making in river basin systems. Research report: LSI-07-6-R, Universitat Politècnica de Catalunya, Barcelona, Spain, January 2007.

- Rendón-Sallard T., Sànchez-Marrè M., Aulinas M. and Comas J., Designing a Multi-Agent system to simulate scenarios for decision-making in river basin systems. 9th International meeting of the Catalan association of Artificial Intelligence. Perpignan, Francia, October 2006.

- Rendón-Sallard T., Sànchez-Marrè M., Devesa F., and Poch M., Simulating scenarios for decision-making in river basin systems through a Multi-Agent system. 3rd Biennial meeting of the International Environmental Modelling and Software Society, iEMS's 2006, Vermont, USA, July 2006.

- Rendón-Sallard T. and Sànchez-Marrè M., A review on Multi-Agent Systems platforms and Environmental Decision Support Systems simulation tools. Research report: LSI-06-4-R, Universitat Politècnica de Catalunya, Barcelona, Spain, January 2006.

References

[Bratman 1987] Bratman M., Intention, Plans, and Practical Reason. Harvard University Press. Cambridge, MA, USA. 1987.

[Braubach et al., 2004] Braubach, Alexander Pokahr, Winfried Lamersdorf. *Jadex: A Short Overview*, in: Main Conference Net.ObjectDays, AgentExpo (2004).

[Dam and Winikoff, 2003] Dam K. H. and Winikoff M., *Comparing Agent-Oriented Methodologies*, In Proc. of the Fifth Int. Bi-Conference Workshop at AAMAS03, (2003).

[Dam, 2003] Dam, K.H., *Evaluating and Comparing Agent-Oriented Software Engineering Methodologies*. Master thesis for RMIT University, Australia (2003).

[Davis, 1983] Davis R. and Rich C., Experts Systems: Part 1-Fundamentals. Tutorial No. 4. The third National Conference on Artificial Intelligence. Menlo Park, CA: American Association for Artificial Intelligence.

[Devesa, 2006] Devesa F., *Desenvolupament d'un Sistema de Suport a la Decisió Ambiental per a la Gestió de les Infraestructures Hidrauliques, amb l'Objectiu de Garantir la Qualitat de l'Aigua a la Conca del Besòs.* Phd Thesis for University of Girona, Spain (2006).

[Fergurson, 1992] Ferguson I. A., Towards an architecture for adaptive, rational, mobile agents. In Werner E. and Demazeau Y., editors, Decentralized AI 3 – Proceedings of the Third European Workshop on Modeling Autonomous Agents and Multi Agent Worlds (MAAMAW-91), pg. 249-262, Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, 1992

[Finin et al, 1997] Finin T., Labrou Y. and Mayfield J., KQML as an agent communication language, in Bradshaw J., Software Agents, MIT Press, Cambridge,1997

[Fox and Das, 2000] Fox, J. and Das, S. *Safe and sound. Artificial Intelligence in Hazardous Applications*, AAAI Press / The MIT Press , 2000.

[Funtowicz and Ravetz, 1993] Funtowicz, S.O. and Ravetz, J. R.: Science for the post-normal age, Futures 25(7), 739-755, 1993

[Funtowicz and Ravetz, 1999] Funtowicz, S.O. and Ravetz, J.R. Post-Normal Science – an insight nom maturing, Futures 31(7), 641-646, 1999

[Genesereth, 1991] Genesereth M. R., Knowledge Interchange Format, In Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, 1991

[Georgeff et al, 1987] Georgeff M. P. and Lansky A. L., Reactive reasoning and planning. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pg. 677-682, Seattle, WA,1987

[Gonzalez and Dankel, 1993] Gonzalez A. and Dankel D. *The Engineering of knowledge-based systems. Theory and practice*. Prentice Hall (1993)

[Gorry, 1989] Gorry G.A. and Scott M., A Framework for Management Information Systems. Sloan Management Review (1989)

[Huhns and Stephens, 1999] Huhns, M. N., and L. M. Stephens. 1999. Multi-agent systems and societies of agents. pages 79-120 *in*: Weiss, G., ed. Multi-agent systems. MIT Press. Cambridge, Massachusetts.

[Iglesias et al, 1998] Iglesias C. A., Garijo M., Gonzalez J. C., A Survey of Agent-Oriented Methodologies, In Proceedings of the Workshop on Agent Theories, Architectures and Languages, France, 1998

[Jadex Tutorial] L. Braubach, A. Pokahr, and A. Walczak. Jadex Tutorial. 2005.

[Jennings, 2001] Jennings N. R., Faratin P., Lomuscio A. R., Parsons S., Sierra C. and Wooldridge M., Automated negotiation: prospects, methods and challenges, International Journal of Group Decision and Negotiation, 2001

[Little, 1970] Little J.D., Models and Managers: The Concept of a Decision Calculus. Management Science. (1970)

[Mangina, 2002] Mangina E., *Review of software products for Multi-Agent Systems. Applied Intelligence* (UK) Ltd for AgentLink: http://www.AgentLink.org/ (2002).

[Metcalf and Hedí, 2003] Metcalf and Hedí, 2003. Wastewater Engineering. Treatment and Reuse. McGraw Hill. 4th edition.

[Nwana et al, 1999] Nwana S. H., Agents: An Overview, The Knowledge Engineering Review Vol. 11, No 3, pp. 1-40, Cambridge University

[Padgham and Winikoff, 2004] Padgham L. and Winikoff M.,*Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and sons (2004).

[Poch et al, 2004] Poch M., Comas J., Rodríguez-Roda I., Sànchez-Marrè M., Cortés U., *Designing and building real environmental decision support systems*. Environmental Modelling and Software, 19 (9), 857 - 873. ISSN: 1364-8152 (2004).

[Pokahr, 2005] Pokahr A., Braubach L., Lamersdorf W.. *Jadex: A BDI Reasoning Engine*, Chapter of Multi-Agent Programming, Kluwer Book, Editors: R. Bordini, M. Dastani, J. Dix and A. Seghrouchni (2005).

[Rao and Georgeff 1995] Rao, A. and Georgeff, M., BDI Agents: from theory to practice. V. Lesser. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95). The MIT Press. Cambridge, MA, USA. 1995.

[Rendón-Sallard et al, 2006] Rendón-Sallard T., Sànchez-Marrè M., Devesa F. and Poch M. *Simulating scenarios for decision-making in river basin systems through a Multi-Agent system*. In Proc. of the 3rd Biennial meeting of the International Environmental Modelling and Software Society IEMSs 2006 (2006).

[Sànchez-Marrè et al, 2008] Sànchez-Marrè M., Gibert K., Sojda R.S., Steyer J.P., Struss P. Rodrigues-Roda I., Comas J., Brilhante V., and Roehl E. A. Environmental Modelling, Software and Decision Support. Elsevier Science (2008)

[Schank and Abelson, 1977] Schank R., and Abelson R., Scripts, plans, goals and understand-ing. Lawrence Erlbaum, 1977.

[Schank, 1982] R. Schank. Dynamic memory: a theory of learning in computers and people. Cambridge University Press, 1982.

[Simon, 1960] Simon H. A., The New Science of Management Decision. New York: Harper & Row. 1960.

[Steels, 1990] L. Steels. Components of expertise. AI Magazine 11(2):28-49, 1990.

[Sudeikat et al, 2004] Sudeikat J., Braubach L., Pokahr A., and Lamersdorf W., *Evaluation of Agent-Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform*, in: Workshop on Agent-Oriented Software Engineering, AOSE (2004).

[Weiss, 1999] Prologue: multi-agent systems and distributed artificial intelligence. Pages 1-23 in: Weiss, G., ed. Multi-agent systems: a modern approach to distributed artificial intelligence. MIT Press. Cambridge, Massachusetts.

[Wooldridge et al., 1995] Wooldridge, M. and N. R. Jennings. 1995. Intelligent agents: theory and practice. Knowledge Engineering Review 10(2):115-152.

[Wooldridge et al., 2000] Wooldridge M., Jennings N. R., and Kinny D., The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, 2000