DERIVING PROJECT-VALUES OF MULTI-AGENT SYSTEM

DISSERTATION

Submitted in partial fulfillment

of the requirements for the Degree of

MASTER OF TECHNOLOGY (M.Tech)

in

Information Technology

by

Vijayananda D Mohire

Roll No: 921DMTE0113

Karnataka State
Open University

Information Technology Department

Karnataka State Open University

Manasagangotri, Mysore – 570006

Karnataka, India

2011

(Submitted June 2011)

DEDICATED

TO

MY BELOVED PARENTS

AND

MY BENIGNANT SUPERVISORS

The development of a Dissertation is a legitimate combination of inspiration, guidance, perspiration and teamwork. It lasts as a novel reminiscence, in the pursuit of career, and in the proliferation of one's imagination.

DISSERTATION APPROVAL

*Dissertation entitled "**Deriving Project values of Multi-Agent System**" by Vijayananda D Mohire is approved for the Degree of Master of Technology in Information Technology*

Examiners

External Supervisor
(Mr. Hriskesh Majumdar)

Internal Supervisor
(Mr. Shivakumar Gowda K)

Chairman

Date

Place *Bangalore*

COURSE WORKSHEET

Subject Code	Subject Name	Credits
	SEM I	
MT 11	Interactive Computer Graphics	4
MT 12	Advanced Computer Architecture	4
MT 13	Algorithm Analysis and Design	4
MT 14	Advanced DBMS	4
	SEM II	
MT 21	OO Software Engg. With UML	4
MT 22	AI and Neural Networks	4
MT 23C	E-Commerce, M-Commerce and Network Security	4
MT 24C	Network Programming	4
	SEM III	
MT 31C	Wireless and Mobile Networks	4
MT 32C	ERP and CRM	4
MT 33	Dissertation- Part I	8
	SEM IV	
MT 41	Dissertation- Part II	16
	TOTAL CREDITS	**64**

CERTIFICATE

This is to certify that the Dissertation entitled ***"Deriving Project-values of Multi-Agent System"****, submitted by Vijayananda D Mohire having Roll Number 921DMTE0113 for the partial fulfilment of the requirements of Master of Technology in Information Technology Degree of Karnataka State Open University (KSOU), Mysore, embodies the bonafide work done by him under my supervision.*

Place *Bangalore*

Signature of the External Supervisor

Date

Name
(Mr. Hriskesh Majumdar)

Designation
(Manager – IT)

Employing organization name
(HP India, Bangalore)

Karnataka State Open University

	:	*Karnataka State Open University, Mysore, India*
Roll No.	:	*921DMTE0113*
Name of the student	:	*Vijayananda D Mohire*
Student's email address	:	*vijay_mohire@yahoo.com*
Internal supervisor's name	:	*Mr. Shivakumar Gowda K*
Internal supervisor's Employing organization	:	*Tejas Network, Bangalore, India*
Internal supervisor's email address	:	*kb_shiva@yahoo.com*
Dissertation title	:	*Deriving Project-values of Multi-Agent System*
Signature of the student, with date and place	:	
Signature of the internal supervisor, with date and place	:	

ABSTRACT

The purpose of this dissertation is to explore and demonstrate the values of Artificial Intelligence based Multi Agent System (MAS) and its benefits for the IT Team which intents to venture and invest its time and resource for project development.

Successful delivery of IT projects is a growing concern among most IT vendors. Despite the availability of the right resources and budget still many IT projects overrun their stipulated time and budget. It has been estimated that on an average 80% of the projects encounter some type of issue either in terms of time overrun, project budget overrun or reduced success rate. This dissertation focuses on ways to assess the pitfalls faced in Artificial Intelligence (AI) based project implementation in general and more specific to Multi Agent system, their remedies and the right mix of the solution to deliver the IT project on time and within budget. This dissertation will try to produce sufficient evidence in order to generate sufficient confidence prior to venturing into MAS projects. This dissertation will try to research the above mentioned topic through an extensive study of relevant literature, case studies, market surveys and the implementation of practical research to gain insights of the actual working Multi Agent System.

This research expects to produce a number of key findings: a) historical evidences to confirm a significant increase in the complexity in developing MAS projects b) the different types of methodologies used in recent years to resolve such complexities, systematic practical approaches in reaching the project targets supported by facts and figures, and c) case studies and examples; a global solution to address the common problems in MAS project implementation.

The main conclusion to be drawn from this research is that current approaches to MAS project are deficient because they fail to embrace a holistic approach instead opting for a narrow tool-based focus. These researches argues for a right approach and customised model to suite MAS specific development and not blindly use the traditional Object Oriented approaches of Analysis and Design (OOAD) of project development. One that leverages industry standards, frameworks and policies and builds a custom model, resolving implementation issues (Agent specific know how's and best practices), and the need for global solution.

Contributions from this study can help IT vendors enhance their capabilities by leveraging the clear, concise and precise details on Multi Agent system, key facts and figures related to project development and reusable content and references from wide range of research scholars, academicians and Industry leaders. Suitable examples used in this study can be leveraged in order to understand the detailed aspects and working know how of the intricacies of the Multi Agent system technology. I am confident that the IT project team will understand and agree upon the paradigm shift in project development as related to Artificial intelligence and will acknowledge the need to focus their attention in gaining the crucial differences and the radically different approach while dealing with Artificial Intelligence based projects.

Broad Area of Work *Artificial Intelligence*

Key words *Agent, Agent Communication Language, Agent development frameworks, Agent-oriented modelling, Agent oriented Software engineering, Agent features and benefits, agentTool III, Amineplatform, Case studies, Design and Development of Multi Agent System, INGENIAS, JADE, JADE Test Suite Addon , Measuring Project value, Project metrics*

ACKNOWLEDGMENTS

*I express my deep sense of gratitude to our **Head of IT Department**, KSOU for giving me an opportunity to work on this Dissertation. I am greatly indebted to him for his cheerful disposition, guidance, advice as and when we approached him.*

*I am grateful to the Study center, Infysys academy, Bangalore in making this a big success. Many thanks for the timely help and attention in making this Dissertation possible within specified timeframe. Special thanks to **Mr. Vivek** and **Mr. Prakash** for their timely help and guidance.*

*I thank my supervisor, **Mr. Shivakumar Gowda K,** who has made a sincere attempt to extend his help and cooperation. Despite of his full time dedication at Tejas Networks, Bangalore, he has devoted his valuable time and has been instrumental in providing his expertise guidance and technical acumen in making this Dissertation a success.*

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ACL	–	*Agent Communication Language*
AI	–	*Artificial Intelligence*
AOSE	–	*Agent oriented software engineering*
API	–	*Application programmer's interface*
BDI	–	*Belief, Desire, and Intention*
DB	–	*Database*
DSL	–	*Domain Specific Language*
FIPA	–	*The foundation for intelligent, physical agents*
ISO	–	*International standards organization*
IT	–	*Information Technology*
J2EE	–	*Java 2 Enterprise Edition*
JADE	–	*Java Agent Development Framework*
JDK	–	*Java Development Kit*
JRE	–	*Java Runtime Environment*
MAS	–	*Multi Agent System*
M.Tech	–	*Masters of Technology*
OOAD	–	*Object Oriented Analysis and Design*
R&D	–	*Research and Development*
SDLC	–	*Software Development Life cycle*
UI	–	*User Interface*

DERIVING PROJECT-VALUES OF MULTI-AGENT SYSTEM

Chapter 1

Introduction

The main inspiration in choosing the topic "***Deriving Project-values of Multi-Agent System"*** was the problem I faced when I set out to work on a project related to AI and in particular Multi Agent System (abbreviated as MAS). I had tried to find a single document/book that provided me the overall picture of using MAS right that explained me the advantages of MAS in a layman's language, simple and not specific to a single technology or platform. I was able to gather information on agent development using Java and few other technologies. However I was not satisfied by the information available in the Market today and I failed to obtain clear, concise and precise information on building MAS.

1.1 Statement of the Problem

The problem I am going to address in this Dissertation is that the challenge faced by an IT team in deciding to go for a MAS development.

This is a common reoccurring problem for every single project in MAS which needs a systematic work from scratch. The problem becomes more serious when a contract is signed off and the delivery team is not confident on Where, How and When to start and manage the MAS project. The IT team is faced with a situation convincing client as to why they need a MAS system when a conventional system will serve the purpose.

1.2 Background and Need

The traditional approach to Project development and learning has, for millennia, rested on the central premise that for projects to be implemented the project had to be awarded or an opportunity window had to be in hand. Only then the IT staff will be ready to stretch their legs and start thinking about the ways to go about the Project. However as time progressed and the need for quick turnarounds, project work has made life more competitive and miserable. Also the use of custom methodology and non standardised process had made the project life cycle look like a long road ahead. Situations have worsened as project sponsors set limits on time, budget and resources there has ever been a great demand for Rapid development and delivery of IT projects. The need for standardised process and implementation methodology has been in great need to control project expenses yet deliver a quality product.

Such situation worsens as there are more available resources and the ever increasing market competition that drives businesses out of their core. Due to the rapid development of tools and technologies especially the "**IT Boom**" has created enormous amount of opportunities and research work has been growing at fast pace. To catch the every growing IT needs, projects have to be more realistic and it applies to the areas of Artificial Intelligence too. Artificial Intelligence cannot remain an R&D concept and it need not be restricted to advanced scientific projects only. Artificial Intelligence needs to be utilised in the commercial applications for day to day domestic purpose.

Michael Wooldridge said in one of his papers (Wooldridge, 1997) published in 1997: "*Unless researchers recognize that agent-based systems are about computer science and software engineering more than they are about AI, then within a decade, we may well be asking why agent technology suffered the same fate as so many other AI ideas that seemed good in principle*."

Kephart and Chess in their article published in 2003 (Chess, 2003) titled "The Vision of Autonomic Computing" states: *Among the applications of tomorrow, much of them will be biologically inspired: self-organising sensors networks, allowing the control of aerospace vehicles, or of dangerous zones; but also storage facilities, or operating systems facilities, which, like the human nervous system, controls in a transparent way significant functionalities*

Looking at the above quotes, it is clear that there is a need for clear understanding and better value creation of the AI and MAS into real life. This Synopsis will attempt to find the true value of MAS project and the pros and cons that are for and against the utilization of MAS in commercial projects. I will try to throw light on the untouched areas of value creation by use of MAS.

1.2.1 Aims of the study

The aim of this dissertation is in ***identifying and evaluating the value generated by incorporating a MAS project.*** It focuses mainly on comparing the MAS based tools, **project building blocks, identifying key features that generate value and rich returns,** and lays a strong foundation that can help the project team to gain confidence and gain significant insight involved in developing the MAS. It can be considered as a baseline reference while taking up MAS based projects and defining the features, benefits and values.

The following objectives have been identified of paramount importance in helping to achieve the aforementioned aim:

1.2.2 Objectives of the study

The major objectives of this Dissertation are

1. Identify the current trends in project development related to MAS
2. Explore the Traditional and MAS project development features
3. Critically Evaluate the benefits of MAS
4. Explore Industry frameworks, standards and best practices
5. Formulate solution recommendations

1.2.3 Rationale

The rationale underlying this synopsis is the larger problem being faced by IT Project team and the IT Industry at large. The major concerns are that as of day, teams are not confident in taking up MAS projects and delivering them in time and budget. Also the IT team is unaware of the intricacies. This Dissertation will attempt to bridge this gap and prove the need of this study.

1.3 Purpose of the Project

Based on the above provided background information, I am undertaking this study to provide a concise, precise and practical approach and knowledge that can be leveraged in the MAS based projects. I will try to provide the necessary justification by use of metrics, data and case studies in taking up MAS projects.

1.4 Research Questions

The major Research questions that will be tested and examined in this study are:

1. What are the forces driving MAS project and the barriers to the successful delivery of projects?
2. What methodologies, tools, models and frameworks are available to support IT staff in coping with MAS projects and how helpful are they?
3. What all can the IT staff leverage during MAS project implementation, testing and management?
4. What all metrics, facts and figures can the end project use in order to showcase the value delivered?

Chapter 2

Review of Literature

After conducting a thorough research on the Internet and web sites, I am able to provide the below earlier history on issues faced during implementation of AI projects. These provide me an insight to the Issues and Challenges faced by the Researcher and Industry and what they did to overcome these.

2.1 Literature review One

Historical Attempts – Frankenstein (Shelley, 1818)

The original story, published by Mary Shelley, in 1818, describes the attempt of a true scientist, Victor Frankenstein, to create life, Figure 2.1. The main challenges faced here was "How can a mere human anatomy can be interfaced electrically to generate desired output like movement, speech etc."

Figure 2.1 Frankenstein creates the fiend - illustration by Bernie Wrightson

2.2 Literature review Two

Wikipedia research: (Anonymous, 2001), (Anonymous, History of artificial intelligence, 2011)

In the history of artificial intelligence, an AI winter is a period of reduced funding and interest in artificial intelligence research. The process of hype, disappointment and funding cuts are common in many emerging technologies (consider the railway mania or the dot-com bubble), but the problem has been particularly acute for AI. The pattern has occurred many times:

- 1966: the failure of machine translation,
- 1970: the abandonment of connectionism,
- 1971−75: DARPA's frustration with the Speech Understanding Research program at Carnegie Mellon University,
- 1973: the large decrease in AI research in the United Kingdom in response to the Lighthill report,
- 1973−74: DARPA's cutbacks to academic AI research in general,
- 1987: the collapse of the Lisp machine market,
- 1988: the cancellation of new spending on AI by the Strategic Computing Initiative,
- 1993: expert systems slowly reaching the bottom,
- 1990s: the quiet disappearance of the fifth-generation computer project's original goals

The worst times for AI were 1974−80 and 1987−93

The first AI winter 1974−1980: In the 70s, AI was subject to critiques and financial setbacks. AI researchers had failed to appreciate the difficulty of the problems they faced. Their tremendous optimism had raised expectations impossibly high, and when the promised results failed to materialize, funding for AI disappeared. At the same time, the field of neural nets was shut down almost completely for 10 years by Marvin Minsky's devastating criticism of perceptrons.

The second AI winter 1987−1993: The first indication of a change in weather was the sudden collapse of the market for specialized AI hardware in 1987. Desktop computers gaining speed and power and in 1987 became powerful than the more expensive Lisp machines. There was no longer a good reason to buy them. An entire industry worth half a billion dollars was demolished overnight.

Chapter 3

Research Methodology

3.1 Research Methodology

Research methodology refers to an organised approach to problem-solving that includes (1) collecting data, (2) formulating a hypothesis or proposition, (3) testing the hypothesis, (4) interpreting results, and (5) stating conclusions that can later be evaluated independently by others.

The above points are realized by use of suitable techniques in this synopsis. These are detailed in the Table 3.1.

Table 3.1 Research Methodology

Step	Method	Technique used
1	Collecting data	• Conduct a research of the Internet, Academic websites, Case studies, Journals and Books to obtain key information about the current trends and the state of the MAS based projects. • Gather metrics related to traditional development and Multi Agent development to define the values • Gather traditional and MAS project concepts, processes and methodologies to derive the values. • Document and Analyse the above collected data
2	Formulating a hypothesis or proposition	• After gathering and analysing data, arrive at the hypothesis about the present MAS project issues, need for better insight while starting the MAS project
3	Testing the hypothesis	• Using suitable tools conduct tests to demonstrate that the value proposition being provided by MAS and how these values can be leveraged for the benefits of the project.
4	Interpreting results	• Tabulate the observations from the Theory, Tools and Case studies and compare those using suitable parameters.
5	Stating conclusions	• Using the results, derive the conclusion about the study undertaken and how this can help the project team.

The details of the above techniques will be covered in the "Report on the Present Investigation" Chapter. The key answers to the major Research questions defined earlier are resolved in Table 3.2:

Table 3.2 Resolutions to Research Questions

Sl.No.	Question	Resolution
1	What are the forces driving MAS project	• A MAS **distributes computational resources and capabilities** across a network of interconnected agents. Whereas a centralized system may be plagued by resource limitations, performance bottlenecks, or critical failures, an MAS is decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems. • An MAS allows for the **interconnection and interoperation** of multiple existing legacy systems. By building an agent wrapper around such systems, they can be incorporated into an agent society. • An MAS **models problems** in terms of autonomous interacting component-agents, which is proving to be a more natural way of representing task allocation, team planning, user preferences, open environments, and so on. • An MAS efficiently **retrieves, filters, and globally coordinates** information from sources that are spatially and temporally distributed.

Sl.No	Question	Resolutions
2	Barriers to the successful delivery of projects	• **Immaturity of Technology**. The agent research community has produced multiple languages, methods and tools for engineering agent-based systems. Although for research purposes diversity is bliss, it hampers industrial adoption. In particular, lack of a common understanding of key multi-agent concepts, a common set of notations and models, and flexible, industrial strength methods and techniques for developing multi-agent systems, has a major negative effect on adoption. • **Lack of Integration** Industrial systems are complex, diverse, and commonly integrated from multiple components. Thus, when introducing agent-based solutions into such systems, it is necessary to integrate with existing software environments such as legacy systems, frameworks, services, etc. It is however difficult to perform such integration and fully maintain agents' autonomy behavior encapsulation. • **Research Bias** towards Scientific Challenges. Yet another issue hindering industrial adoption is the bias of scientific research towards problems of scientific challenge. Multi-agent research tends to adopt scientific models and languages which are not always known to, or understood by, software developers. The multi-agent research community commonly delivers rather complex concepts and tools, not always accessible to the typical software designer. • **Limited Scope of Programming Languages**. Although agent-oriented programming is widely studied, practical work on the software engineering aspects of agent programming languages is limited in scope. That is, scientific research on agent-oriented programming usually does not address programming language design issues.

Sl.No	Question	Resolutions
3	What methodologies, tools, models and frameworks are available to support IT staff in coping with MAS projects and how helpful are they?	Tools are discusses in details in the Tools evaluation section and the process, methods and framework they use are detailed. These will provide an insight into the usage of the tools for effective project development.

Sl.No	Question	Resolutions
4	What all can the IT staff leverage during MAS project implementation, testing and management?	Details of each stage of the SDLC and the Management activities, role of each Project member (RASIC charts) and all the project artifacts like Document, Code, Deliverables has been discussed in following sections A clear usage of the tools has be demonstrated along with their usage benefits

Sl.No	Question	Resolutions
5	What all metrics, facts and figures can the end project use in order to showcase the value delivered?	These data has been discussed in Metrics section. Suitable results and facts have been discussed in the Results section.

3.2 Modules

The modules that are going to be used are demonstrated in following sections.

Agent specific modules:

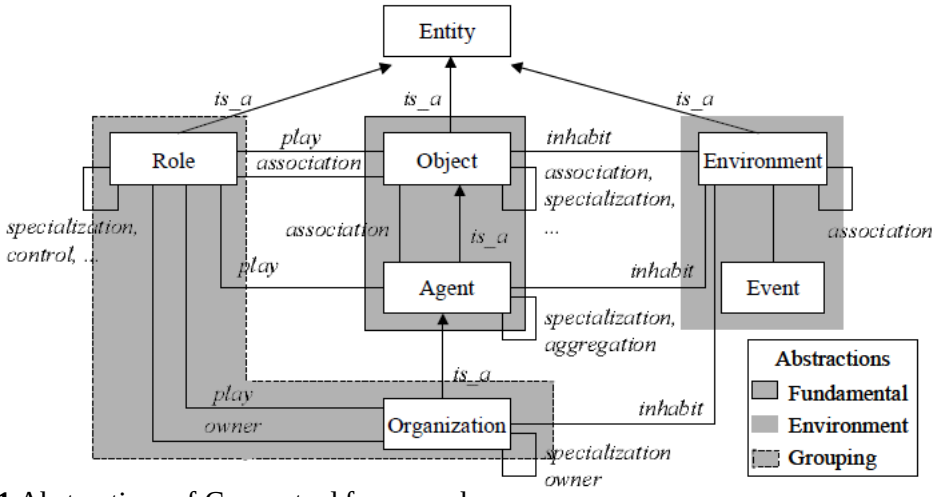


Figure 3.1 Abstractions of Conceptual framework

The modules depicted in Figure 3.1 are based on the famous TAO classification. (Torres, 2002) TAO classifies the set of abstractions it defines into three distinct categories: (i) *fundamental abstractions* - include the object and agent abstractions, which are the basis for building MASs (ii) *environment abstractions* - include definition of environments and events that are used to represent the environmental constraints and characteristics that influence instances of fundamental and grouping abstractions; (iii) *grouping abstractions* – encompass abstractions for dealing with more complex situations in large-scale systems; it includes organizations and roles to model complex collaborations.

Fig. 3.1 presents the most important abstractions of TAO and their relationships. There are a total of 8 abstractions including entity, object, agent, organization, environment, event, object role and agent role. It is advised for the MAS project to include the above classifications in their Framework to ensure proper working and communication among the agents. Their relations should be normalised to ensure atomicity and avoiding duplicate relations.

3.3 Data Structure

OBJECT: An object has control of its state. It performs operation that can modify its state during its lifetime. On the other hand, an object cannot modify its behavior1 and has no control of it, i.e., an object is not autonomous in the sense it does everything that another one asks it to do. In this way, objects are passive entities that do whatever anyone asks them to do and only when they are asked. A common object class defines the structure and behavior of similar objects. The following template, Figure 3.2 defines the object class.

---------- **Object** ----------

Object_Class Object_Class_Name
State setof {Information}
Behaviour setof {Operation}
Relationships setof {Relationship_Name}
end Object_Class

Figure 3.2 Object template

AGENT: The agent template, Figure 3.3 defines an agent *class*. An agent *class* describes beliefs, goals, actions, plans and relationships that are the same for all agent instances of it.

---------- **Agent** ----------

Agent_Class Agent_Class_Name
Beliefs setOf {Belief_Name}
Actions setOf {Action_Name}
Plans setOf {Plan_Name}
Relationships setOf {Relationship_Class_Name}
end Agent_Class

Figure 3.3 Agent template

ENVIRONMENT: The environment template, Figure 3.4 presents an environment *class*. An environment class defines its state as a set of resources and a set of services, the behaviour of its instance as a set of its properties and a set of relationships that are common to all environment instances.

---------- **Environment** ----------

Environment_Class Environment_Class_Name
Resources setOf {< Resource, Permission, Entity_Class_Name >}
Services setOf {< Service, Permission, Entity_Class_Name >}
Behaviour setOf {Properties}
Relationship setOf {Relationship_Name}
end Environment_Class

Figure 3.4 Environment template

RELATIONSHIP: The relationship template, Figure 3.5 is used to define the links between the entities. For each relationship type, the template identifies the entities and its roles in the relationship.

---------- **Environment** ----------

Relationship Relationship_Name
INHABIT: habitat, citizen
OWNERSHP: owner, member
INHERITANCE: super-entity, sub-entity
PLAY: entity, role
CONTROL: controller, controlled, condition
DEPENDENCY : client, supplier, condition
ASSOCIATION: Entity_Class_Name1, Entity_Class_Name2
AGGREGATION: aggregator, part
end Relationship

Figure 3.5 Relationship template

3.4 Implementation

Implementation methodology followed for MAS project is Iteration process and JADE methodology. In brief this methodology is suitable as it helps to iterate progressively and to incorporate changes and generate matured results, instead of last minute surprises at the end when many integration issues can crop up.

3.5 Reports

Below text provides the List of reports that are generated for this dissertation. The reports are specific to the Tools being used and are made available in Tool specific evaluation section later in this document. Each phase of development needs different information and these are captured as reports.

1. Modelling reports
2. Design reports
3. Implementation reports
4. Testing reports
5. Others- Admin, Management etc.

3.6 Network Architecture

Typical network architecture for MAS is shown in Figure 3.6. The main entities of this are 1) Browser 2) Agent platform front end 3) Agent Container. Browser hosts the Applet Container that interacts with the other entities through Java RMI method invocation. Agent platform front end hosts the necessary services like Directory services, Agent Management services. Agent container hosts the agents.

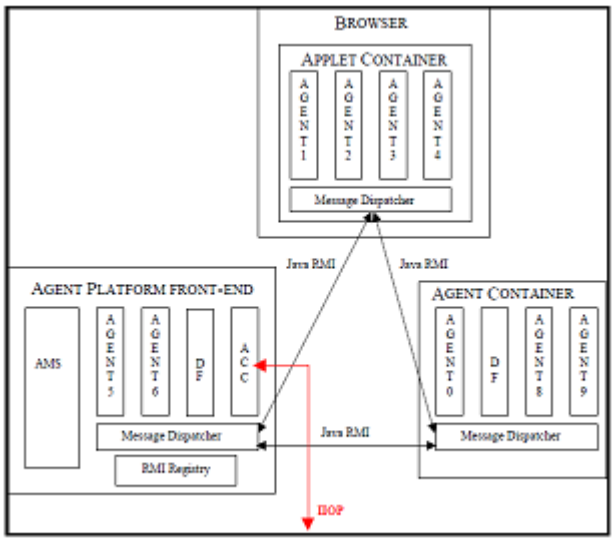


Figure 3.6 Software Architecture of one JADE Agent Platform

3.7 Security

Implementation of security mechanism at various levels is implemented using JADE Security based on Java platform. Typical structure of the mechanism is shown Figure 3.7

Figure 3.7 Structure of a FIPA ACL Message

An ACL message is composed of two parts: the **envelope**, which contains transport related information and the **payload**, which contains the actual sensitive data.

The JADE authentication mechanism is based on the JAAS (Java Authentication and Authorization Service, http://java.sun.com/products/jaas) API that enables the enforcement of differentiated access control on system users. Features like Authentication, Authorisation and Encryption of messages are leveraged in the MAS system.

3.8 Tools / Platform, Hardware

Table 3.3 shows the tools and platform used for this project. It mainly uses the Open Source based tools and based on Java, Prolog, XML Ontology and Agent specific Software Engineering.

Table 3.3 Tools / Platform

Sl.No	Tools / SW applications	Purpose/ Details
1	INGENIAS Development kit(IDK)	IDK is a framework for analysis, design and implementation of multi-agent systems (MAS). It is based on the specification of MAS meta-models, from which tools such as the model editor and code generation are generated
2	Amineplatform	Amine is a Multi-Layer Java Open Source Platform dedicated to the development of various kinds of Intelligent Systems (Knowledge-Based, Ontology-Based, Conceptual Graph Based, NLP, etc.) and Intelligent Agents.
3	Packet-World	The Packet-World is a 100% Java test bed for investigating situated multiagent systems. The tool allows users to experiment with various aspects of multiagent systems, incl. agent architectures, stigmergy, communication, etc
4	agentTool III (or aT3)	agentTool is a Java-based graphical development environment to help users analyze, design, and implement multiagent systems. It is designed to support the new Organization-based Multiagent Systems Engineering (O-MaSE) methodology.
5	JADE	Support for Middleware and hosting Java Agents compliant to FIPA standards

Chapter 4

Report on the Present Investigation

4.1 Evaluation methods

Table 4.1 Evaluation methods

Evaluation method	Inputs	Process	Expected Outputs
Theory assisted	Historical Theories related to Agent technology	My observations and interpretation that generate the results.	Data depicted in form of suitability of the Theory and its merits. Results section provides the outputs
Tool assisted	Different tools that can be used for the projects	My observations that lead to the results	Generate an overall comparison matrix of the tools considering key factors
Case studies assisted	Case studies that depict the key needs of the society and public utility	Observation that lead to the results	Usefulness of the MAS to particular situation

4.2 Evaluation methods: Theory assisted

Multi-agent systems have to be plausible from a philosophical, psychological, and Mathematical point of view. For each of these points of view, alternative theories exist and the various theories are discussed in Table 4.2. **Philosophically**, the organization can be seen from the viewpoints of realism and constructivism. **Psychologically**, several agent types can be distinguished on the way they think, and behave. **Mathematically** agents can be studied under different empirical formulas and frameworks

Table 4.2 Evaluation Methods- Theory

Sl.No	Theory	Theory details
1	Philosophical theory (René, 1993)	Important propositions in the philosophy of AI include: Turing's "polite convention": *If a machine acts as intelligently as a human being, then it is as intelligent as a human being.* The Dartmouth proposal: *"Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it."* Newell and Simon's physical symbol system hypothesis: *"A physical symbol system has the necessary and sufficient means of general intelligent action* Searle's strong AI hypothesis: *"The appropriately programmed computer with the right inputs and outputs would thereby have a mind in exactly the same sense human beings have minds*

Sl.No	Theory	Theory details
2	Psychological theory (René, 1993)	The view that the brain possesses and processes information, which is the principal characteristic of **cognitive psychology,** can be traced back at least to the works of William James (1842-1910). Another scholar Craik specified the three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action. He clearly explained why this was a good design for an agent: If the organism carries a "small-scale model" of external reality and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilize the knowledge of past events in dealing with the present and future, and in every way to react in a much fuller, safer, and more competent manner to the emergencies which face it. Aristotle's approach (with a few minor refinements) was implemented 2300 years later by Newell and Simon in their GPS program, about which they write (Newell and Simon, 1972): The main methods of GPS jointly embody the heuristic **of means-ends analysis.** This kind of analysis—classifying things in terms of the functions they serve and oscillating among ends, a function required, and means that performs them—forms the basic system of heuristic of GPS. Means-ends analysis is useful, but does not say what to do when several actions will achieve the goal, or when no action will completely achieve it.

Sl.No	Theory	Theory details
3	Mathematical theory (René, 1993)	In 1930, Kurt Godel (1906-1978) showed that there exists an effective procedure to prove any true statement in the first-order logic of Frege and Russell; but first-order logic could not capture the principle of mathematical induction needed to characterize the natural numbers. In 1931, he showed that real INCOMPLETENESS limits do exist. His **incompleteness theorem** showed that in any language expressive enough to describe the properties of the natural numbers, there are true statements that are undecidable: their truth cannot be established by any algorithm **Decision theory,** pioneered by John Von Neumann and Oskar Morgenstern (1944), combines probability theory with utility theory (which provides a formal and complete framework for specifying the preferences of an agent) to give the first general theory that can distinguish good actions from bad ones. **Game theory**: The Italian Gerolamo Cardano (1501-1576) first framed the idea of probability, describing it in terms of the possible outcomes of gambling events. Before his time, the outcomes of gambling games were seen as the will of the gods rather than the whim of chance, Probability quickly became an invaluable part of all the quantitative sciences, helping to deal with uncertain measurements and incomplete theories

4.3 Metrics and Comparisons of Agents

The MAS system project development will be compared, contrasted and suitable metrics that are needed to prove these will be used to generate the needed proof to showcase the differences between the traditional project development and Agent specific project developments. Relevant references to traditional process, methods and metrics will be used and how these are compared to the Agent specific development will be highlighted. Metrics used here are shown in Table 4.3

Table 4.3 Agent comparison metrics

Sl.No	Comparison unit / Metrics
1	Project metrics like SDLC, Time, cost of investment, ROI, Features benefits etc
2	Nature of control and command like – Autonomy, Grained access control, security mechanism
3	Behaviour of agents and its benefits
4	Communication protocols, Error handling mechanism
5	Problem solving skills- how close it solves the real problems and complexity
6	Classical examples, case studies, social implications, trends
7	Application specific metrics like Response time, Error handling, customisation etc

Below section tries to bring about the contrasting features and differences between Agent and Object technology

4.3.1 Abstraction:

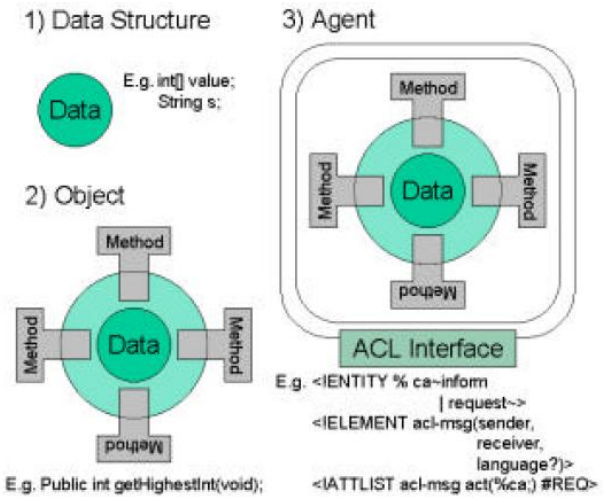


Figure 4.1 Abstraction levels

Figure 4.1 shows the specifications at each level of abstractions. In 1) a user-created data structure is defined, 2) methods are added to allow manipulation of the underlying data, giving us an object, 3) to create an agent the object(s) are wrapped in an ACL interface that specifies how to interact with the agent in this case via a DTD(Document Type Definition)

Object-oriented (OO) programming is programming where data-abstraction is achieved by users defining their own data-structures (see figure 1), or "objects". These objects encapsulate data and methods for operating on that data; and the OO framework allows new objects to be created that inherit the properties (both data and methods) of existing objects. The agent-oriented (AO) approach pressures the developer to think about objects as agents that make requests of each other, and then grant those requests based upon who has made the request. Agent systems have been developed that rely purely on the inherited network of accessibility of OO systems (Binder 2000), but ideally an AO programming environment would provide more fine-grained access/security control through an ACL (Agent Communication Language) interface.

Given that an objective may be specified (e.g. transfer data X from A to B), Autonomy can be thought of as the ability of an entity to revise that objective, or the method by which it will achieve that objective, and an ***Agent*** is an Object that possesses ***Autonomy***.

Below Figure 4.2 shows few differences between the Object and Agent Abstractions based on Structural elements and Relations.

	OOP	AOP
Structural Elements		
	abstract class	generic role
	class	domain specific role
	member variable	knowledge,belief
	method	capability
Relations		
	collaboration (`uses`)	negotioation
	composition (`has`)	holonic agents
	inheritance (`is`)	role multiplicity
	instantiation	domain specific role + individual knowledge
	polymorphism	service matchmaking

Figure 4.2 Key differences between OOP and AOP

4.3.2 Behaviour:

Agents may also employ some degree of nondeterministic (or unpredictable) behaviour. When observed from the environment, an agent can range from being totally predictable to completely unpredictable. Usually, object classes are designed to be predictable in order to facilitate interaction. Agents are commonly designed to determine their behaviour based on individual goals and states, as well as the states of ongoing conversations with other agents

.

4.3.3 Communication:

Communication in **Object Oriented Systems** happens as follows:

- Sender sends message
- Receiver executes method
- Receiver has (almost) no control over whether it executes method
- Behaves like servant

Communication in **Agent Oriented Systems** happens as follows:

- Agents don't consider invocation of methods
- Agents are autonomous, meaning they have control over internal state and execution of behavior
- Consider agents i and j where i has capability of performing action α
- j wishes for i to perform α
- j sends a message to i to request it to perform α
- i makes a local decision about whether to perform α based on
  - Local interests
  - Trust
  - Agreement…

4.3.4 Problem Solving

Although object-orientation is an outstanding approach for software development, yet as professional software development has gained experience in this paradigm, it seems obvious that the principles of object-orientation are limited to a certain degree in developing complex systems which are based on the synergy between human and computers. The techniques and modeling mechanisms provided by the object orientation are not fully adequate for supporting human centered modeling process because:

- They focus on structural problem-directed approaches which represent views of situations in terms of object configurations, property values and associations. Thus, user's perspective or context related issues are not explicitly or thoroughly considered.

- In practice, it is quite often this paradigm conflicts with the way people describe their work or responsibilities. According to (Glass, 1996, Kawalek, 1995), people describe their work through activities, tasks and goals rather than objects

- Agent-oriented approach promotes a societal view of computation. An agent possesses knowledge and methods and the ability to engage in complex communication with other agents, including human agents, in order to obtain information or request their help in accomplishing its goals and tasks. Like objects, agents interact with each other via message passing. However, each message is also defined in terms of mental activities. An agent may engage another (or itself) with messaging activities from a predefined class of messages protocol (Parks, 1998). The class of message protocol are: informing, requesting, offering, accepting, rejecting, competing, and assisting.

Taking into account that the aforementioned features promised by agent-orientation, it seems to have many strengths but also some weaknesses, for example:

- Agent-oriented approach focuses primarily on tasks (sometimes referred to as activities or goals). It encourages too much attention to be paid on the detailed functions and too little to the main structure of the business problem domain or the structure of the data/components used for accomplishing a task.

- As being rather a premature method, the agent-oriented analysis and design methods are not fully developed. It is rather obvious that the principles of agent-orientation rely on representing mental states to model agent interaction. Thus this paradigm is an appropriate medium to support goal/task-based modelling.

4.3.5 Metrics:

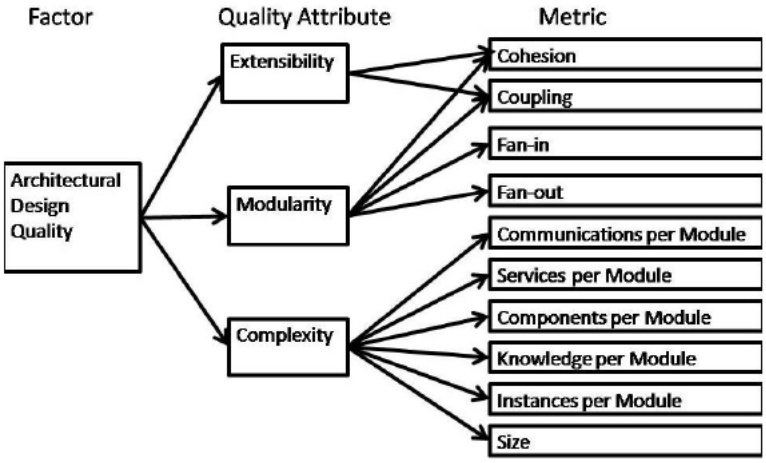


Figure 4.3 Measuring quality attributes of agent oriented architectures

It is worth noting that this work takes inspiration from the **factors-criteria-metrics** approach proposed by McCall. According to that (see Figure 4.3), some metrics are proposed to assess quality attributes of the system.

Metrics for extensibility

The extensibility metrics evaluates whether an agent-oriented architecture is designed to include hooks and mechanisms for expanding/enhancing the architecture with new capabilities without having to make major changes to the infrastructure of the architecture. The dependencies between components need to be detected for applying the presented metrics. In object-oriented architectures, dependencies can consider: data type coupling, data coupling, control coupling and content coupling. Table 4.4 matches these dependencies with agent-oriented dependencies, inspired by the following similarities. In object oriented architectures, some modules can share data types (i.e. data type coupling); which is equivalent to several kinds of agents that share some languages (i.e. types of data exchanged). These languages are usually expressed in terms of ontologies to be shared (i.e. ontology sharing).

Table 4.4 Kinds of coupling in both object-oriented and agent-oriented architectures

Object-oriented concepts		Agent-oriented concepts		
Name	Description	Name	Description	
Data Type coupling	Two modules use the same data type	Ontology Sharing	Two agents share knowledge for communicating	*Necessary*
Data coupling	Data from one module is used in another	Knowledge coupling	An agent uses knowledge (facts instantiated from the ontology) of another agent	⇓
Control coupling	One module may control actions of another	Behavioral coupling	An agent can control the behavior of other	⇓
Content coupling	A module refers to the internals of another module	Inner Structural Coupling	An agent uses internal elements of another agent	*Against MAS principles*

Metrics for modularity

Modularity is the degree to which a MAS component may be separated and recombined. Modularity can depend on many variables in agent-oriented architectures. Some of these variables are the Cohesion and Coupling metrics, and the Fan- in and Fan-out metrics, which are defined in Table 4.5

Table 4.5 Metrics for the Modularity in agent-oriented architectures

	Name	Definition
Fi	Fan-in	The number of incoming communication dependencies of a module.
Fo	Fan-out	The number of outgoing communication dependencies of a module.

Metrics for complexity

Complexity characterizes MAS with many parts in intricate arrangement. Inspired by the complexity measure in traditional software engineering, this work presents several metrics with regards to the relationships among agent oriented architectural elements, amounts of elements per module (i.e. knowledge elements, components and instances) and the global amount of elements (i.e. size). These metrics are defined in Table 4.6

Table 4.6 Metrics for Complexity in agent-oriented architectures

	Name	Definition
ACmM	Average of Communications per Module	The number of communication protocols divided by the number of modules.
ASM	Average of Services per Module	The number of services divided by the number of modules.
AKM	Average of Knowledge per Module	The number of knowledge elements divided by the number of modules.
ACM	Average of Components per Module	The number of components of the system divided by the number of modules.
Sz	Size	Sum of the number of modeling elements of each kind. If necessary, the sum can be weighted by their kinds.
AIM	Average of Instances per Module	The average of instances of each module in the deployment.

4.4 Error handling:

The agent architecture contains four main elements to correspond with the execution model, namely the perception, actuation, internal mechanisms, and internal representation. (Platon, 2007). These elements are depicted in Figure 4.4.

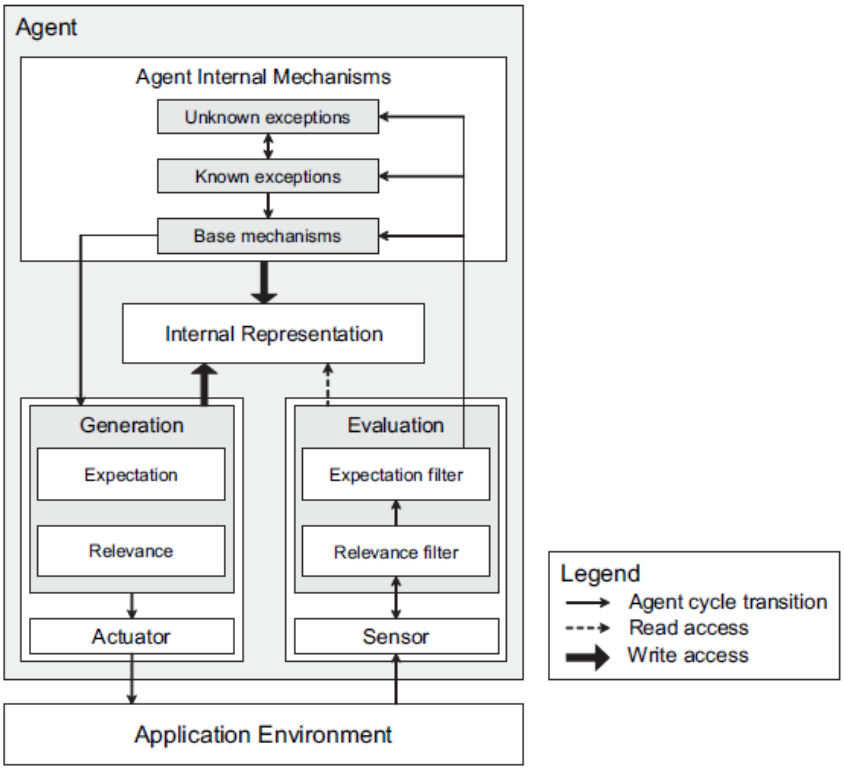


Figure 4.4 Agent base architecture for exception management

Perception/Sensor: The perception element encompasses the sensory functionalities and the evaluation function. Sensors receive and interpret events from the environment and pass them to the evaluation. This latter element is responsible for estimating the relevance and the appropriateness of the events. An event is categorized as relevant by the relevance filter when it pertains to the agent, its acquaintances, or its activities, according to the decision of the agent. The event is otherwise discarded as irrelevant, and the agent then returns to the sensor function. Relevance filters are dynamically generated by the relevance generation in the actuation element to steer the agent perception strategy (similar to the "focus" of active perception .

Relevant events are further evaluated for appropriateness by the **expectation filter** to distinguish expected events from known and unknown exceptions. Awaited events are defined dynamically by the **expectation generation** function in the actuation element, according to the agent decisions.

Agent internal mechanisms & internal representation: An event and its evaluation (expected, known or unknown exception) are forwarded to the agent internal mechanisms. The evaluation uses the internal representation element as reference to distinguish the events by accessing the agent acquaintance network, for example.

The internal representation refers to any representation type inside the agent. For example, the BDI and KGP architectures have a set of knowledge base, whereas some other agents can have simpler internal representations, such as a set of configuration parameters.

The **agent internal mechanisms** element receives evaluated events and activates one of its three elements, depending on the evaluation. Expected events trigger the base mechanisms, whereas exceptions trigger the corresponding mechanisms. The base mechanism can provide facilities such as planners, inference engines, or others such as PRS, MANTA, etc. to deal with expected events. The two exception mechanisms manage the event by setting appropriately the agent internals, so that the base mechanism can handle the case or continue the activities of the agent.

Actuation: The action command finally entails the generation of expectations and relevance criteria for the next evaluation of percepts from the environment by the Evaluation functions. Typically, changing to the next states of an interaction protocol is added as 'relevant expectations' in the internal representation. In the end, the command is applied in the environment by the actuator function.

4.5 Evaluation methods: Tool assisted

Table 4.7 Evaluation methods: Tool assisted

Sl.No	Tool coverage	Associated Tools/Process
1	Design	INGENIAS Dev Kit (IDK), Amine platform
2	Modelling	INGENIAS Model Driven Development(MDD), Amine platform
3	Development	INGENIAS Dev Kit (IDK), Amine platform
4	System engineering	agentTool III based on Multi agent System Engineering (MaSE)
5	Testing	JADE TestSuite Add on, Packet-World
6	Security	Jade Security(Jade-S) , Java authentication and authorization (JAAS)
7	General tools/platforms	Ant, JRE, for developing different Intelligent systems

4.6 Details of Tool assisted evaluation:

4.6.1 INGENIAS Dev Kit (IDK): (Gómez, 2005)

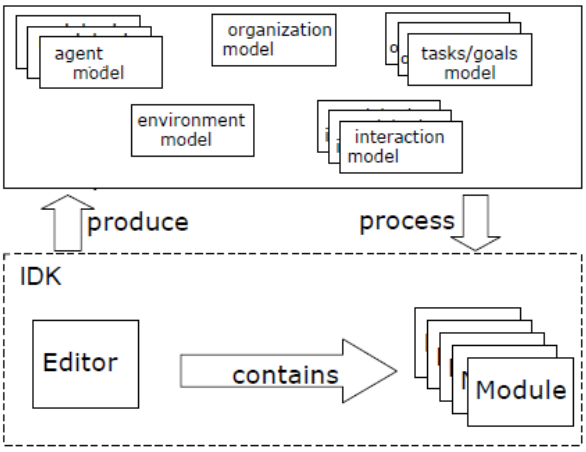


Figure 4.5 Relationship between the IDK, models and components

IDK is based on the definition of a set of meta-models (Figure 4.5) that describe the elements that form MAS from several viewpoints and that allow to define a specification language for MAS using the Rational Unified Process (RUP). The viewpoints are five: agent (definition, control and management of agent mental state), interactions, organization, environment, and goals/tasks.

Refer Fig 4.6 for the five view points of IDK

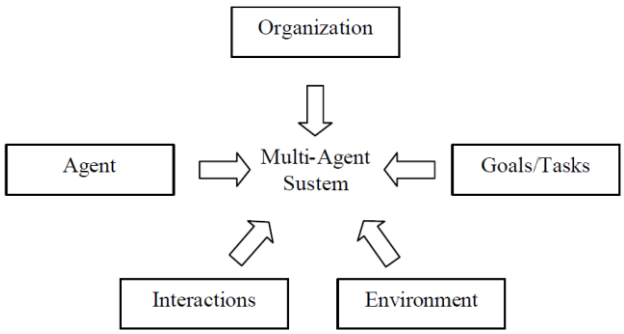


Figure 4.6 MAS specification viewpoints

These viewpoints can be easily realized in RUP process and also can become a part of the Enterprise frameworks like TOGAF or Zachman. This is the first level of architecture; details can be mapped to the Industry standard frameworks for various domains.

4.6.2 Development Process

INGENIAS also defines a development process, the INGENIAS Development Process (IDP). By following IDP, a developer generates the full specification of MAS. To specify the IDP we use activity diagrams that describe the kind of results to obtain. Building each meta-model can be achieved by performing a set of activities in the software development process that leads to the final MAS specification. Initially, activities are organised and represented with UML activity diagrams showing dependencies between them. Instead of showing these activities here, Figure 4.7 summarises the results required in each phase of the Unified Software Development Process.

Meta-models are used as specification language of the MAS the same way as UML does for object oriented applications.

		PHASES		
		Inception	**Elaboration**	**Construction**
WORKFLOWS	**Analysis**	o Generate use cases and identify actions of these use cases with interaction models. o Sketch a system architecture with an organization model. o Generate enviroment models to represent results from requirement gathering stage	o Refined use cases o Agent models that detail elements of the system architecture. o Workflows and tasks in organization models o Models of tasks and goals to highlight control constraints (main goals, goal decomposition) o Refinements of environment model to include new environment elements	o Refinements on existing models to cover use cases
	Design	o Generate prototypes perhaps with rapid application development tool such as ZEUS o Agent Tool.	o Refinements in workflows o Interaction models that show how tasks are executed. o Models of tasks and goals that reflect dependencies and needs identified in workflows and how system goals are achieved o Agent models to show required mental state patterns	o Generate new models o Social relationships that perfect organization behaviour.

Figure 4.7 Results to be obtained in each phase of the development process

4.6.3 IDK Example

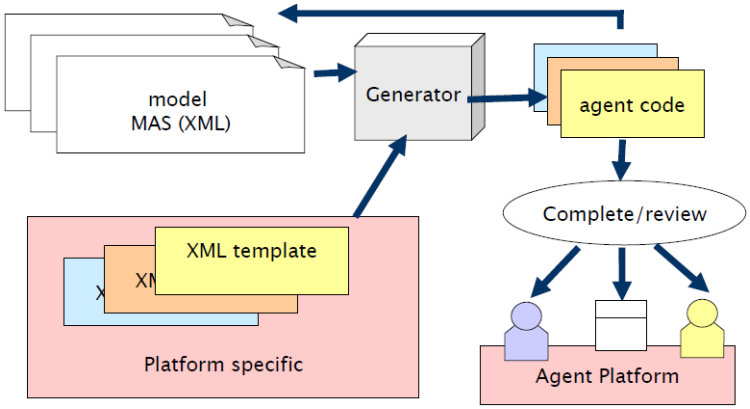


Figure 4.8 IDK Code generation process

Figure 4.8 displays the process of code generation in IDK. We will build a Hello World example to show how the IDK generated code can be deployed in JADE and this can be executed to show the Simple Greeting message. It is assumed that we have the IDK, ANT and JRE environments installed on the PC

STEPS:

1) Move to C:\IDK from console. Type **ant runide**
2) IDK editor is invoked and displayed
3) Choose File -> Load
4) Open the folder IDK\workspace\MyProject\spec
5) Open specification.xml

Create an Agent Definition Diagram

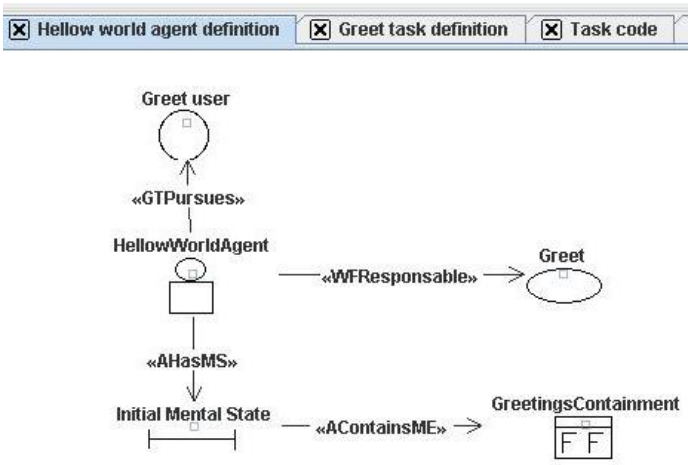


Figure 4.9 Agent Definition

Create a Task\Goal Diagram

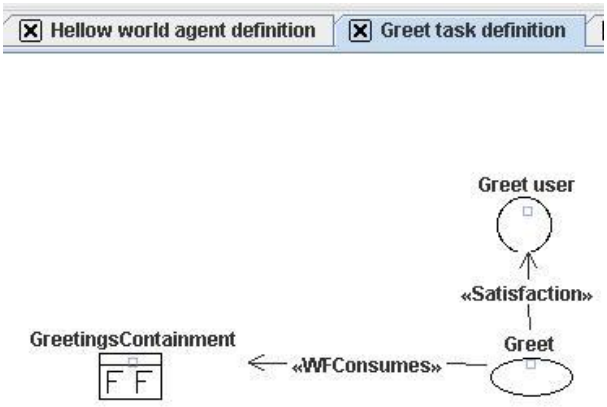


Figure 4.10 Goal Definition

Create a Component Diagram

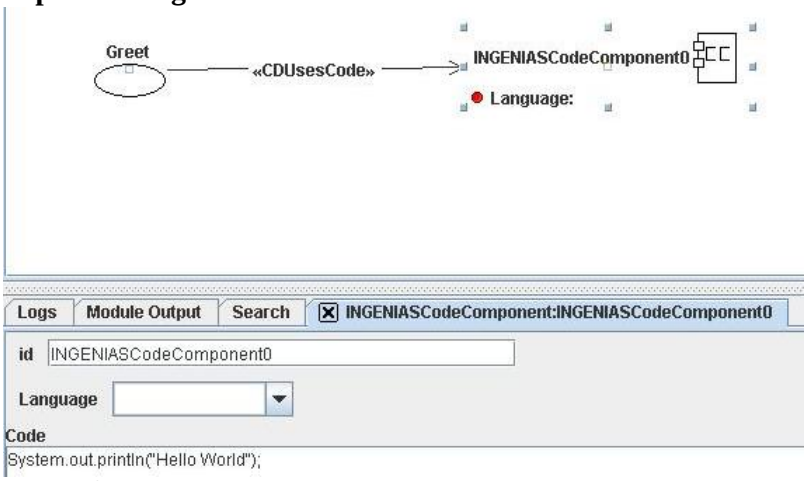


Figure 4.11 Component definition

Go to the menu option Modules -> Code Generator -> Ingenias Agent Framework generator ->Generate

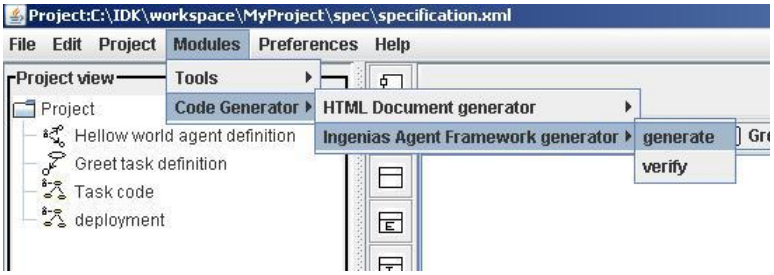


Figure 4.12 Code generation

In my example the code is generated at C:\IDK\workspace\MyProject\gensrc\ingenias\jade\agents. Details of the code generated is provided in Appendix I

Executing the code:

- Go to the project folder (In my example it is C:\IDK\workspace\MyProject)
  - Open two consoles here
- In the first console run
  - **ant runjade**
- Execute this only once for all runs of the system
- In the second one console run
  - **ant run**
- Two GUIs will appear as shown in Figures 4.13 and 4.14

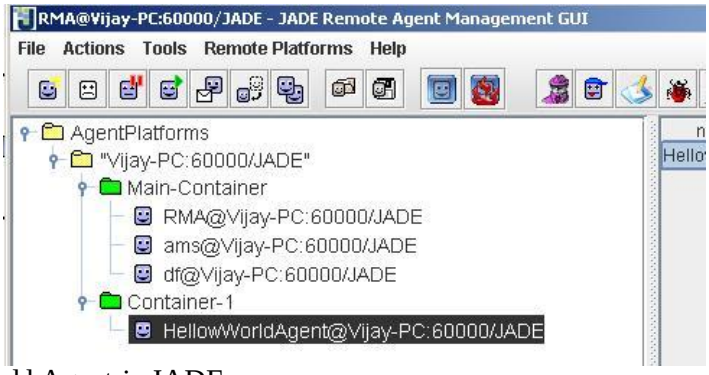


Figure 4.13 Hello World Agent in JADE

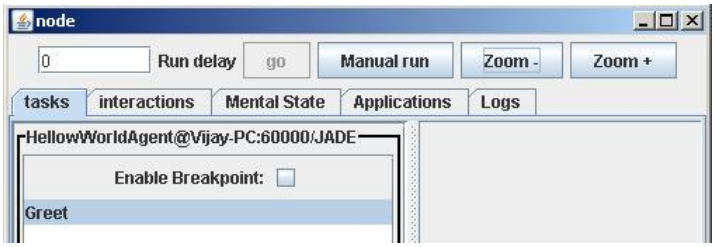


Figure 4.14 Agent Task "Greet" displayed

Invoke the deployment of 3 agents by running: **ant runThreeAgentsDeployment.** Output is:

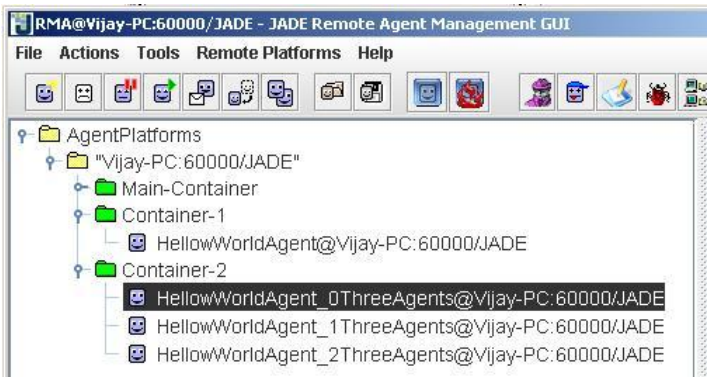


Figure 4.15 Multiple instances of Agents deployed in JADE

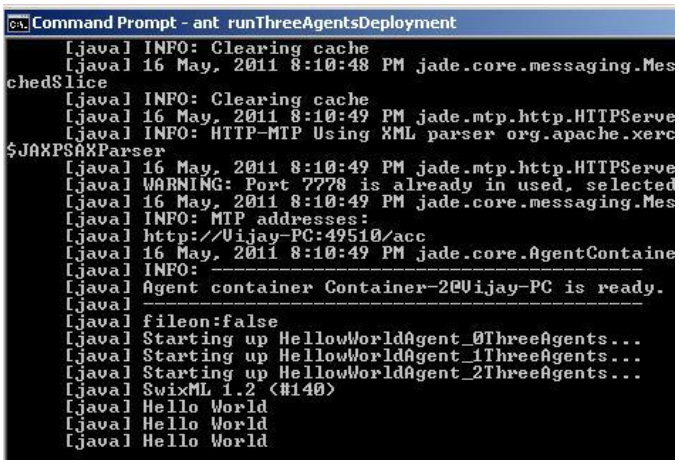


Figure 4.16 Output from Agent displayed

4.6.4 AMINEPLATFORM

Amine is a Java open source multi-layer platform and a modular Integrated Development Environment, dedicated to the development of intelligent systems and multi-agents systems. Amine is a synthesis of 20 years of works, by the author, on the development of tools for various aspects of Conceptual Graph theory (KABBAJ, 2009)

Amine provides also several graphical user interfaces (GUIs): Ontology GUI, Conceptual Graph (CG) Notations editors GUI, CG Operations GUI, Dynamic Ontology GUI, Ontology processes GUI, Prolog+CG GUI and Synergy GUI.

Currently, we use Jade to handle the lower level of the MAS (i.e. creation and communication between agents) and Amine for the higher level (i.e. cognitive and reactive capabilities of the agents are implemented using Amine).

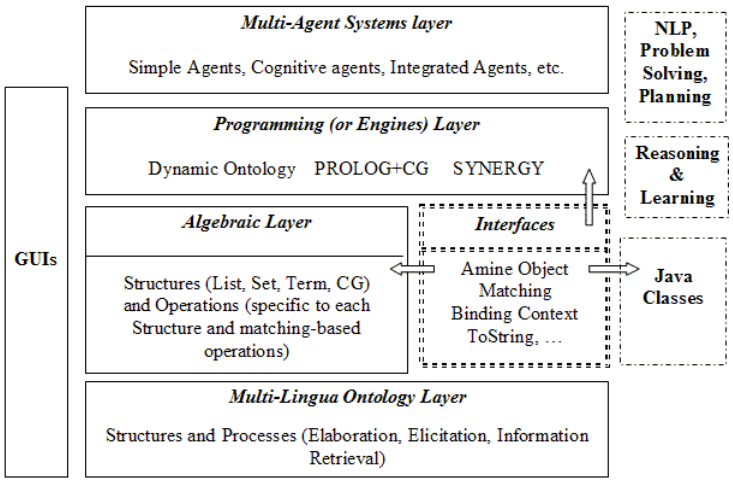


Figure 4.17 Four hierarchical layers of Amine platform

Amine is a modular integrated environment composed of four hierarchical layers: a) ontology layer provides "structures, processes and graphical interfaces" to specify the "conceptual vocabulary" and the semantic of a domain, b) algebraic layer is build on top of the ontology layer: it provides "structures, operations and graphical interfaces" to define and use "conceptual" structures and operations, c) programming layer is build on top of the algebraic layer: it provides "programming paradigms/languages" to define and execute "conceptual" processes and, d) multi-agent layer provides plugs-in to agent development tools, allowing for the development of multi-agent systems. More specifically: (Refer Figure 4.17 above for the 4 layers)

1. **Ontology layer**: It concerns the creation, edition and manipulation of multi-lingua ontology.

2. **Algebraic layer**: this layer provides several types of structures and operations: elementary data types (AmineInteger, AmineDouble, String, Boolean, etc.) and structured types (AmineSet, AmineList, Term, Concept, Relation and Conceptual Graph). In addition to operations that are specific to each kind of structure, Amine provides a set of basic common operations (clear, clone, toString, etc.) and various common matching-based operations (match, equal, unify, subsume, maximalJoin and generalize).

3. **Programming layer**: Three complementary programming paradigms are provided by Amine: a) pattern-matching and rule-based programming paradigm, embedded in PROLOG+CG language which is an object based and CG-based extension of PROLOG language, b) activation and propagation-based programming paradigm, embedded in SYNERGY language, and c) ontology or memory-based programming paradigm which is concerned by incremental and automatic integration of knowledge in an ontology (considered as an agent memory) and by information retrieval, classification and other related ontology/memory-based processes.

4. Agents *and Multi-Agents Systems layer*: Amine can be used in conjunction with a Java Agent Development Environment to develop multi-agents systems. Amine does not provide the basic level for the development of multi-agents systems (i.e. implementation of agents and their communication capabilities using network programming) since this level is already offered by other open source projects (like Jade). Amine provides rather plugs-in that enable its use with these projects in order to develop multi-agents systems

4.6.5 AMINEPLATFORM Example

Intro to example: In this section, we present Renaldo; a MAS that concerns the simulation of a child story. The setting of the story is a forest; it corresponds to the environment of Renaldo. The characters of the story (the bear John, the bird Arthur, the bee Betty, etc.) are the agents of Renaldo. Each type of agents (bear, bird, bee, etc.) has a set of attributes, knowledge, goals, plans and actions that are specified as Prolog+CG programs. A specific agent can have, in addition, specific attributes, knowledge, goals, plans and actions, specified also as Prolog+CG programs

The MAS Renaldo:

The MAS Renaldo is implemented as a Java class RenaldoApplication:

```
public class RenaldoApplication {

public static void main (String [] arguments) {

    // Specification of agent: AgentName: CompleteName_of_Agent_Class (Param1 Param2 Param3 ... ParamN)

    String arg1 = "John:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(John.pcg Environnement.pcg ontology.xml)";

    String arg2 = "Arthur:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(Arthur.pcg Environnement.pcg ontology.xml)";

    // put these arguments in an array

    String [] args = {arg1, arg2};

    // call the constructor of a MAS by specifying its name and the agent arguments array. MAS Renaldo will create two agents: John and Arthur

    new AmineJadeMas ("Renaldo", args);

    // you can retrieve each agent object by calling the static getAgentById method. The Id of the agent is specified as a String

    PPCGAmineJadeAgent John = (PPCGAmineJadeAgent) AmineJadeMas.getAgentById ("John");

    // use this agent object to call the predefined methods to satisfy goal or send messages between agents.

    // In the following case, we assign to john a goal to satisfy that is expressed as a CG

    John.satisfyGoal ("Animal (John, Goal): [Animal: John] <-pat-[SatisfyHungry]-intensityOf-> [Hungry = _Level]");

  }

}
```

The agent John in the MAS Renaldo

We will focus on the agent John. The same principles hold for the other agents like Arthur. The "body" of John is the PPCGAmineJadeAgent class. The "mind" of John is specified by its parameters: Prolog+CG files "john.pcg", "Environment.pcg" and "ontology.xml". These parameters are considered by the Prolog+CG interpreter that is bind to the agent (recall that each agent PPCGAmineJadeAgent has its own Prolog+CG interpreter).

File "Environment.pcg" is a Prolog+CG program that is common to all Renaldo agents and that contains information about the environment, like position of various objects, trajectories, etc.

Goal AskAnimalFriendForFood involves sending messages: agent _Agent will send two messages to his friend _Friend. Let us consider how it is done in detail. Note first that values of _Agent and _Friend will be two Individuals, in this case Individuals John and Arthur. They are specified in the Ontology: John as an Individual of Bear and Arthur as an Individual of Bird. Prolog+CG considers identifier John as a reference to an Individual Conceptual Structure, and the same for Arthur. So, the first task is to get the string name of the Individuals Arthur and John, which are "Arthur" and "John" respectively.

Details of the Renaldo classes and xml files can be found in Appendix II and the enclosed CD. Once the code has been modified for the purpose of seeking the Goal then the batch file can be created to run the code.

Execution of Renaldo MAS

The last step is typing runMASRenaldo.bat from command prompt. This will invoke the Renaldo class and the Prolog code also executes based on the code input parameters.

The activation of the MAS Renaldo, which is implemented as a Java class RenaldoApplication (see above), provides the following outcome: Figure 4.18 shows the frame of the MAS Renaldo. It provides a summary of the creation of and communication between agents in Renaldo MAS. Figure 4.19 shows the frame of the agent John and Figure 4.20 shows the frame for the agent Arthur. Recall that John and Arthur are two agents (implemented as processes or threads) that are concurrent. So the two frames (Figures 4.19 and 4.20) should be read in parallel !

Figure 4.18 MAS Agent Renaldo

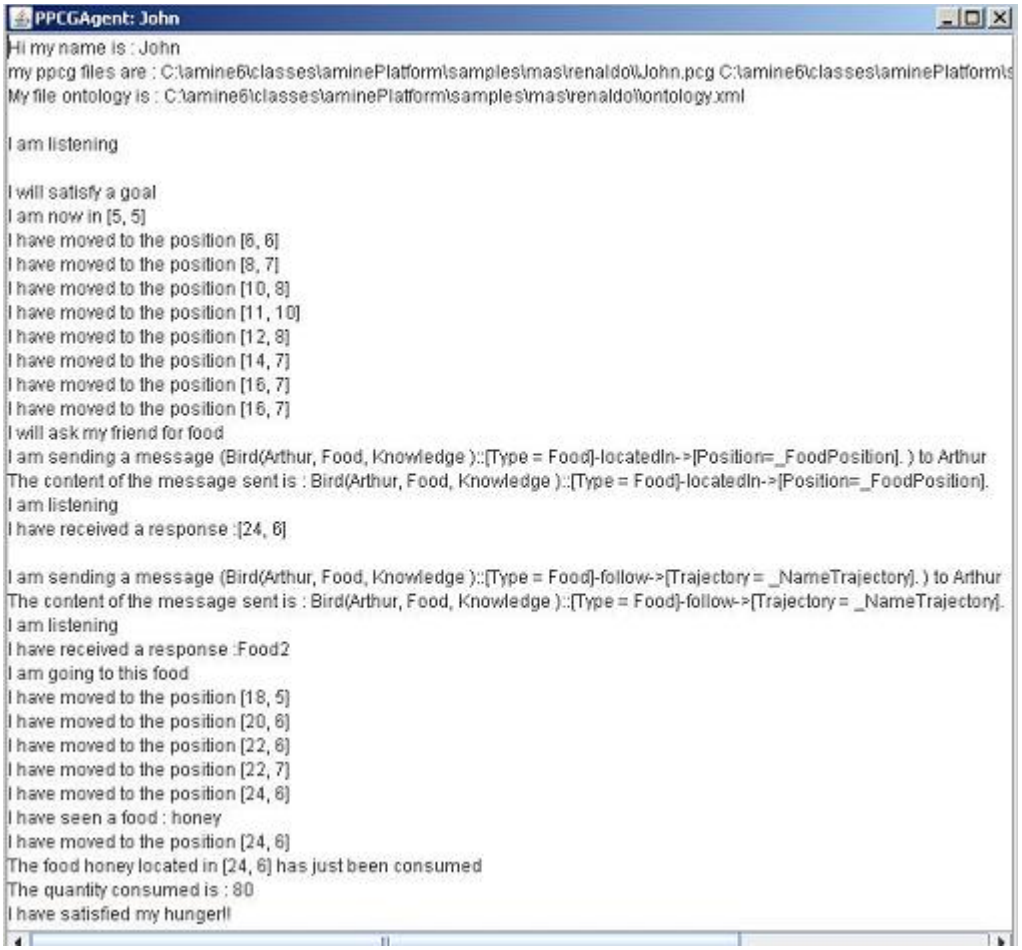


Figure 4.19 Agent John the Bear

Figure 4.20 Agent Arthur the Bird

4.6.6 AgentTool: The overall goal of Agentool is to facilitate the design, verification, and management of customized agent-based O-MaSE (refer Figure 4.8) compliant processes. (Wood, 2000)

Table 4.8 O-MaSE Methods fragments

Work Units		
Activity	**Task**	**Work Products**
Requirement Engineering	Model Goals	Goal Model
	Goal Refinement	Goal Model for Dynamic Systems (GMoDS)
Analysis	Model Organizational Interfaces	Organization Model
	Model Roles	Role Model
	Model Domain	Domain Model
Design	Model Agent Classes	Agent Classes Model
	Model Protocols	Protocol Model
	Model Plans	Agent Plan Model
	Model Policies	Policy Model
	Model Capabilities	Capability Model
	Model Actions	Action Model

The entire MaSE methodology is represented in Figure 4.21.

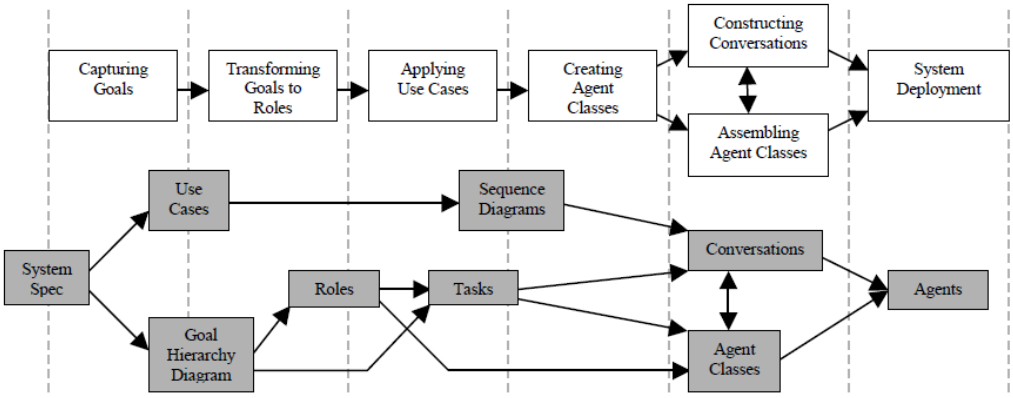


Figure 4.21 MaSE Methodology

Currently agentTool implements three of the seven phases of MaSE, as indicated by the outlined region in Figure 4.22. The planned future of agentTool includes expanding it to cover more of the MaSE methodology. The goal of agentTool is to allow multiagent system designers to formally specify the required structure and behavior of a multiagent system and semi-automatically synthesize multiagent systems that meet those requirements.

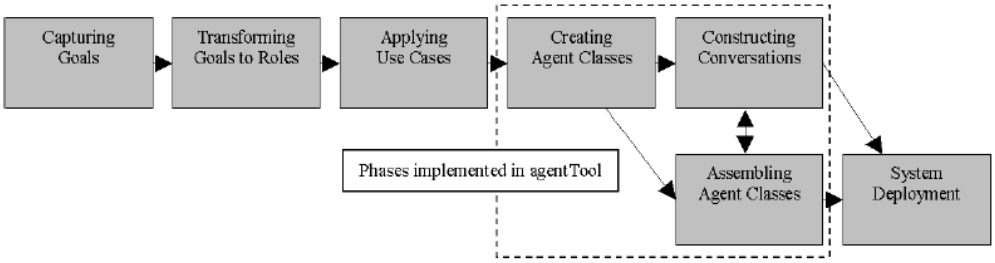


Figure 4.22 MaSE in agentTool

Building a Multiagent System using agentTool

Constructing a multiagent system using agentTool begins at the main system panel, called the Agent Diagram Panel. Different types of diagrams like Goal, Use cases, Sequence, Deployment diagrams can be developed.

4.6.7 Agent Tool Example:

Agent-Poker is an agent that manages the Admission of players for Poker game, Manages the game, and the interfaces for the players who are betting.

Steps: Design the **Goal Hierarchy** for the modules of the game as shown in Figure 4.23.

Figure 4.23 Define Goal Hierarchy

Define the **Use cases** for the Modules / submodules as shown in Figure 4.24

Figure 4.24 Define Use Cases for Poker agent

Define the **Sequence Diagram** for the Use cases as shown in Figure 4.25

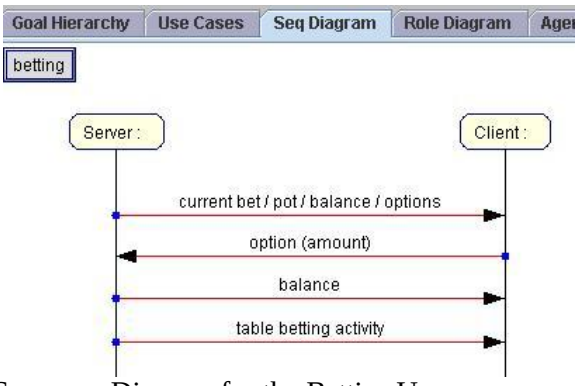


Figure 4.25 Define the Sequence Diagram for the Betting Use case

Define the **Role diagram** (refer Figure 4.26): In this add the Role, Task and Protocol of the Communication.

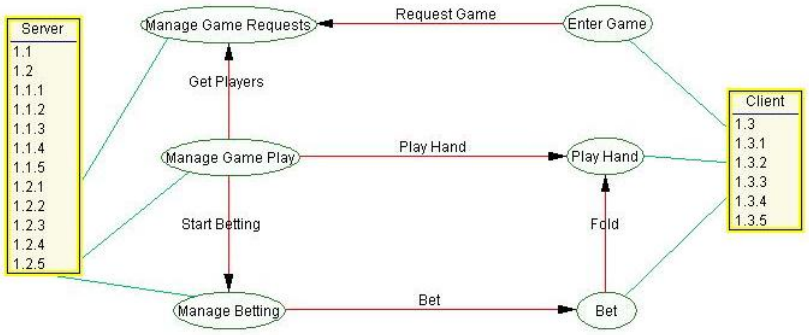


Figure 4.26 Define Role Diagram

Add the **Agent Template Diagram** (refer Figure 4.27) Add the Agents and the Conversations that are depicted by the state diagrams

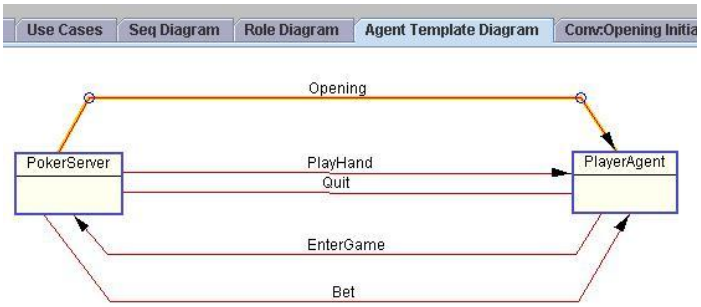


Figure 4.27 Define the Agent Template Diagram

One example of the Conversation **State diagram** for Opening is Conv: OpeningResponsder is shown in Figure 4.28.

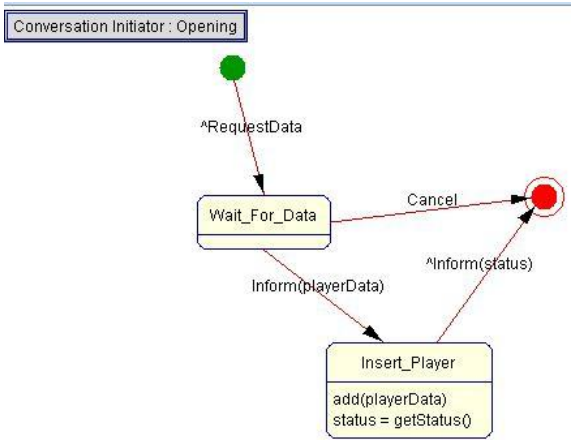


Figure 4.28 Define the State Diagram for Poker Opening

Design the **Deployment diagram** as shown in Figure 4.29 for deploying Agents in JADE

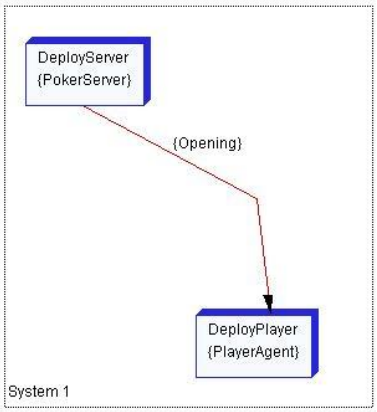


Figure 4.29 Define the System Deployment

Generate the Code: Use the CodeGen- > agentMoM new to generate the Code in a specified directory. (In my case it is C:\AgentTool_1.8.3\Vijayegs\Poker), output shown in Figure 4.30

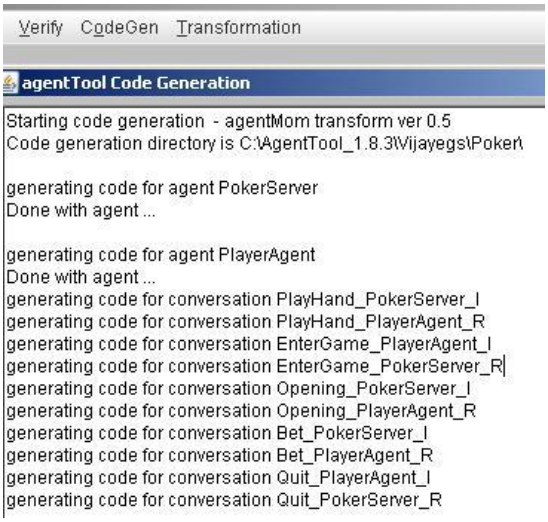


Figure 4.30 Java Code generated for JADE

One sample Code snippet is shown below. Detailed code in Appendix –III

```
/* Class for agent PokerServer
* automatically generated by agentTool - agentMom transform ver. 0.5
*/
package Poker;

import java.net.*;
import java.io.*;
import java.awt.*;
import afit.mom.*;
import java.util.*;

public class PokerServer extends Agent
{
      /* Component Attributes */

/* Constructor for agent.
*  automatically generated by agentTool - agentMom transform ver. 0.5
*/
public PokerServer (String n, int p)
{
      super(n, p);
}
/* Run method for agent.  This must be completed by user.
* automatically generated by agentTool - agentMom transform ver 0.5
*/
public void run()
{
}
/* Write method - used to write conversation error messages to output.
* automatically generated by agentTool - agentMom transform ver 0.5
*/
public void write(String s)
{
      System.out.println(s);
}
/* Required method for agentMom framework - must be completed by user.
* automatically generated by agentTool - agentMom transform ver 0.5
*/
public void receiveMessage(Socket server, ObjectInputStream input,
ObjectOutputStream output)
{
}
}
```

4.6.8 Packet World:

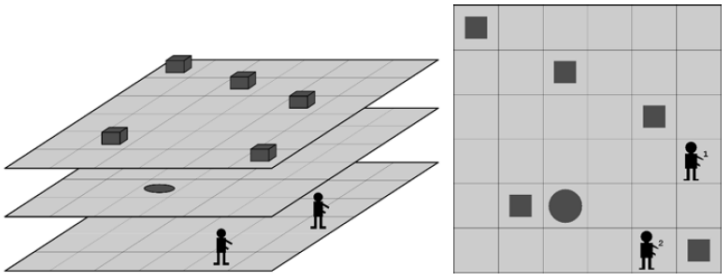
The Packet-World is a test bed for investigating situated MASs, developed in Java. A situated MAS is a computing system composed of a (distributed) environment populated with a set of localized agents that cooperate to solve a complex problem in a decentralized way

.

Situated agents are entities that encapsulate their own behaviour and maintain their own state. They have local access to the environment, i.e. each agent is placed in a local context which it can perceive and in which it can act and interact with other agents. A situated agent does not use long-term planning to decide what action sequence should be executed, but instead it selects actions on the basis of its position, the state of the world it perceives and limited internal state. In other words, situated agents act in the present, "here" and "now".

Intelligence in situated MAS originates from the interactions between the agents, rather than from their individual capabilities. Situated MASs have been applied with success in practical applications over a broad range of domains. Some examples are: manufacturing control, supply chains systems, network support and peer-to-peer systems. The benefits of situated MAS are well known, the most striking being efficiency, robustness and flexibility. (dannyweyns, 2009)

The Packet World is a simulator for situated agents. The environment is composed of a two dimensional grid with packets and destinations(refer Figure 4.31). In this domain robots (agents) must move the packets to the correct destination. The goal of this application is to investigate under which conditions the robots will develop social conventions and how the robots can take advantage of the information communicated to each other.



(a) Environment structure decomposed in layers. (b) Integrated representation in the Packet-World.

Figure 4.31 Layered model of the environment

4.6.9 Packet World Example

From installed location C:\jPakjeswereld, find the different batch files to setup ENV, Size of the TEST World etc. Run the various batch files to configure and test the agent performance.

run.bat run the default application (a menu will appear) (Windows)

runBatch.bat run the application for batch testing (Windows)

runEditor.bat run the world editor (Windows)

runFromFile.bat run the simulator with the configurations from .\configfiles\setup.txt (Windows)

runGui.bat run the simulator (the main application) (Windows)

One sample, runGUi.bat invokes the CONFIG and VIDEO GUI as shown in Figure 4.32 and 4.33

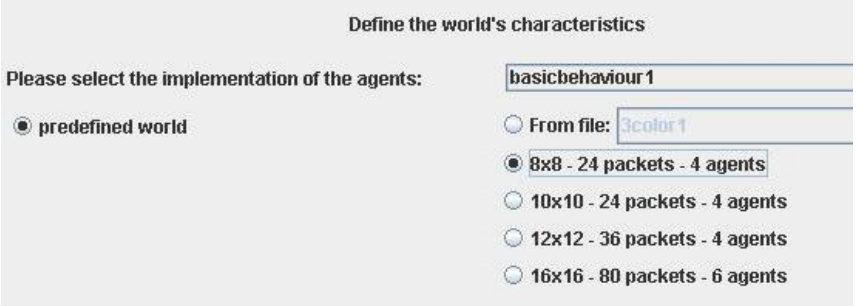


Figure 4.32 Config GUI of PacketWorld

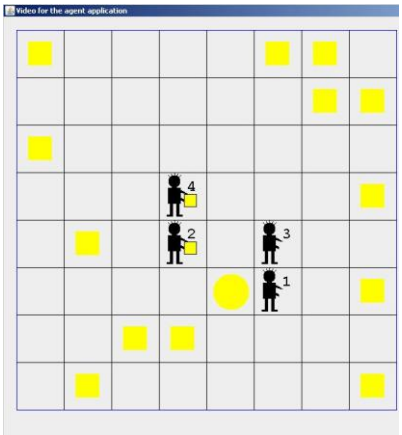


Figure 4.33 Video of the Agents

As shown above the AGENTS are doing their tasks of Picking the Yellow item and aggregating it to single Circle. The computation is displayed in a separate window and hence their Performance can be measured.

4.6.10 JADE

JADE is a software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. The goal is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents

JADE is an agent middle-ware that implements an Agent Platform and a development framework. It deals with all those aspects that are not peculiar of the agent internals and that are independent of the applications, such as message transport, encoding and parsing, or agent life-cycle. (Bellifemine, 2001)

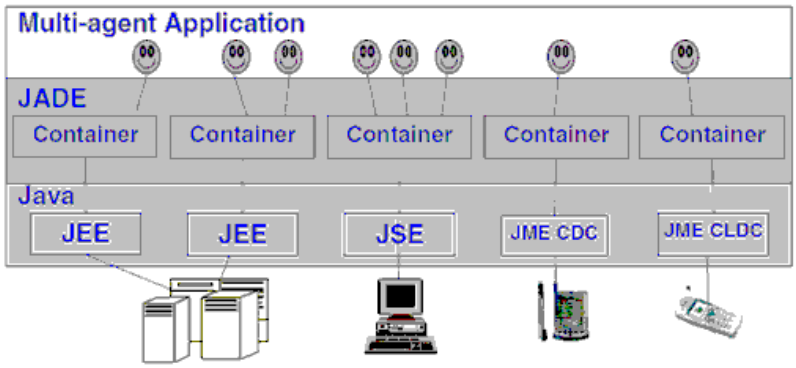


Figure 4.34 The architecture of a JADE agent system

JADE logical view is shown in Figure 4.34. It comprises of Agents developed in Java and deployed into the JADE Containers. These Containers rely on the Java runtime environment on the target platform like Enterprise Servers, Client Server machines, Mobiles etc.

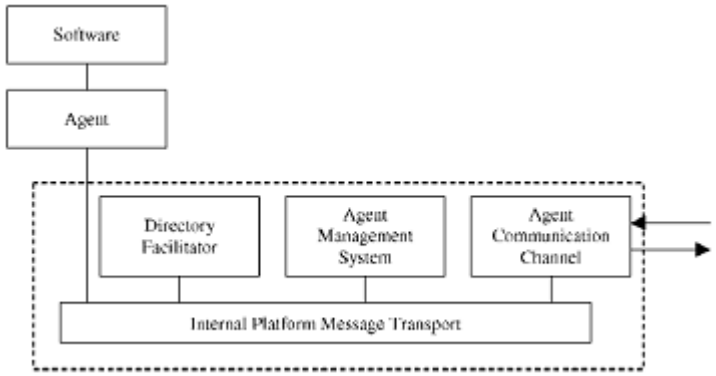


Figure 4.35 FIPA reference model of an agent platform

Based on FIPA standard specifications a reference model has been designed as shown in Figure 4.35. It comprises of Directory facilitator for agent look up services, Agent management system for managing the persistence of agent's data and lifecycle. Agent Communication channel is used for proper interfacing to the external world and cross platform communication. Examples of JADE are covered in other tool examples where the Agents are deployed.

4.6.11 JADE Security:

Authentication

The JADE authentication mechanism is based on the JAAS (Java Authentication and Authorization Service, http://java.sun.com/products/jaas) API that enables the enforcement of differentiated access control on system users. The JAAS mechanism provides a set of de facto LoginModules; the Unix, NT and Kerberos modules (Board, 2005)

The Unix and NT modules are Operating System dependent and are designed to use the identity of the user extracted from the current Operating System session. The Kerberos module is system independent in operation, but requires system-specific configuration prior to use.

Permissions

Reflecting the JADE architecture that includes a Main Container and several peripheral containers, two types of policy files can be used to grant permissions to agents:

1. The MainContainer policy file that specifies platform-wide permissions such as “Agents owned by user Bob can kill agents owned by user Alice”.

2. The peripheral container policy files (one per container) that specify container specific permissions (such as “Agents owned by user Bob can kill agents owned by user Alice on the local container”).

Message integrity and confidentiality

Signature and encryption guarantee a certain level of security when sending an ACL message both to an agent running on the same or a foreign platform. Signatures are a well-known safeguard to ensure the integrity of a message (confidence that data has not been tampered with during transmission) and the identity of the message originator.

Encryption, on the other hand, ensures confidentiality of the message by protecting message data from eavesdropping (confidence that only the intended receiver will be able to read the clear message). As background information, an ACL message is composed of two parts: the envelope, which contains transport related information and the payload, which contains the actual sensitive data.

In JADE-S “Signature” and “Encryption” always apply to the entire payload in order to protect all the important pieces of information contained in the slots of the ACL message (content, protocol, ontology, etc.). The security-related information (such as the signature, the algorithm or the key) is then placed into the envelope

4.6.12 JADE Security Example: I am using the example in my PC at G:\JADE\add-ons\security\examples\startup\main

Authentication: Attaching a container: In this section the necessary steps to attach a container to a JADE agent platform will be shown. For the container, which has to be started, a set of configurations has to be done. The configuration file contains the services that have to be started. services=\

MAIN.CONF file contains the code:

```
# ---- JADE configuration ----
# ------ Platform ------
name=myplatform.example.org

# ------ Services  ------
services=\
jade.core.security.SecurityService;\
jade.core.security.permission.PermissionService;\
jade.core.security.signature.SignatureService;\
jade.core.event.NotificationService

# ------ Agents  ------
agents=alice-rma:jade.tools.rma.rma

# ------ Transport ------
nomtp=true

# ------ Security configuration ------
# ---- Permission ----
# Permission Policy file
java.security.policy=policy.txt

# ---- Authentication ----
# - Type of Prompt -     can be: {Cmdline, Text, Dialog} ('Text' does not
work well with 'ant')
jade.security.authentication.logincallback=Dialog

# - if Cmdline, use this user/pass -
owner=alice:alice

# - Auth module -      can be: {Simple, Unix, NT, Kerberos}
jade.security.authentication.loginmodule=Simple

# - if Simple, use this password file
jade.security.authentication.loginsimplecredfile=passwords.txt
```

Now a policy file, which is specific for this container, has to be created. Here again the permission grants for the JADE source code have to be made explicit. Moreover the full set of permission is granted to user bob. Additionally the generic principal "*" (i.e. everyone) is granted with the right to send message.

POLICY.TXT

```
grant codebase "file:../../../lib/jadeSecurity.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../../../../../lib/jade.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../../../../../lib/jadeTools.jar" {
      permission java.security.AllPermission;  };

// --- Startup example ---
// --- Policy on the MAIN container ---

grant principal jade.security.Name "alice" {
      permission jade.security.PlatformPermission "", "create,kill";
      permission jade.security.ContainerPermission "", "create,kill";
      permission jade.security.AgentPermission "", "create,kill";
      permission jade.security.AgentPermission "", "suspend,resume";
      permission jade.security.AMSPermission   "",
"register,deregister,modify";
      permission jade.security.MessagePermission "", "send-to";
};

grant principal jade.security.Name "bob" {
      permission jade.security.ContainerPermission "container-owner=bob",
"create, kill";
      permission jade.security.AgentPermission "agent-
owner=bob,container-owner=bob,agent-name=bob-*", "create";
      permission jade.security.AgentPermission "agent-owner=bob",
"kill,suspend,resume";
      permission jade.security.AMSPermission   "agent-owner=bob",
"register,deregister,modify";
      permission jade.security.MessagePermission "", "send-to";
};
```

After the above procedure run the **go.bat** file and the above Policies are attached to the Code and Authentication is applied. Figure 4.36 shows the Login dialog shown when run

Figure 4.36 Login Dialog

If you press OK without any entry then the Login fails as shown in Figure 4.37

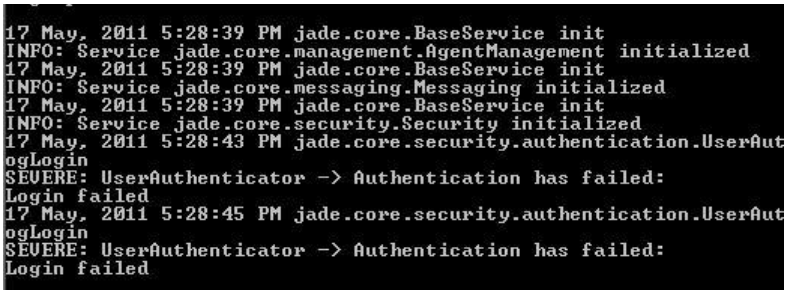


Figure 4.37 Login Failed status

If you provide the correct Login details: alice/alice then the JADE is invoked and the Agent is deployed as shown in Figure 4.38

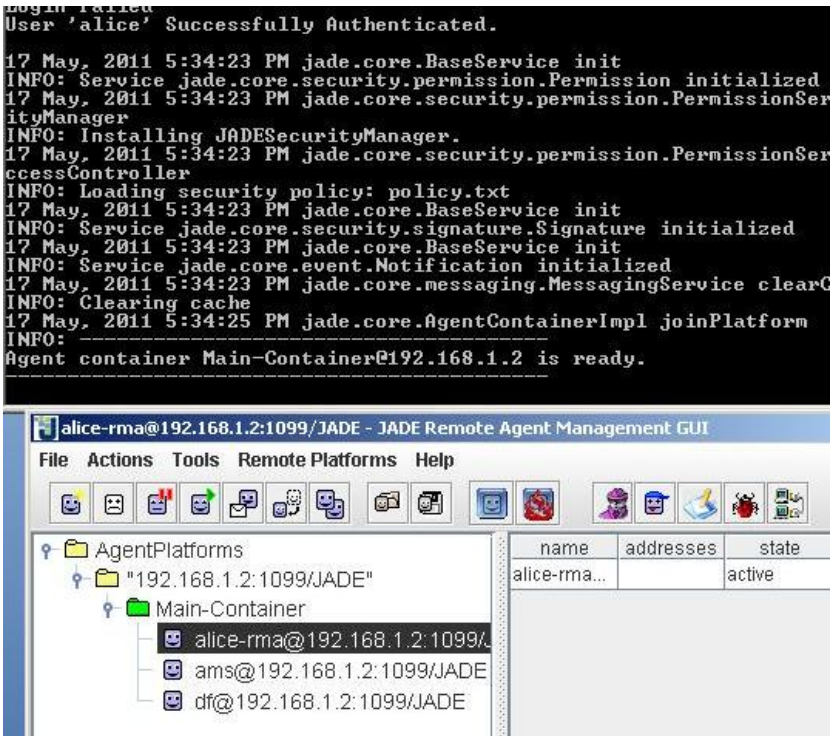


Figure 4.38 Login success status

Authorization

In the illustrated example, user "alice" has full platform wide permissions, but is not granted any permission on container cont-1 (a part from the right of sending messages). As a consequence, using alice-rma it will be possible to create and kill agents owned by alice or bob on the main container. When trying to create an agent on container cont-1 on the other hand, and authorization error should occur and the following dialog window (refer Figure 4.39) should appear.

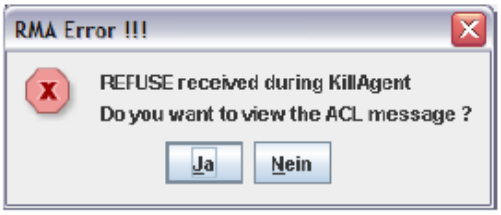


Figure 4.39 Authorisation failure

Similarly, since bob is only granted the right to create agents owned by bob on containers owned by bob and whose name starts with "bob-", trying to create an agent that does not have all these characteristics (e.g. an agent called "a" or an agent on the main container that is owned by alice) will result in the same error.

Exchanging signed and encrypted messages

Since signing and/or encrypting a message clearly slow down the performances of agent communication, messages are neither encrypted nor signed by defaults. It is the responsibility of the agent sending a message to explicitly request the platform to sign and/or encrypt a message.

This is done by means of the **setUseSignature ()** and **setUseEncryption ()** methods of the SecurityHelper that can be retrieved (as for all ServiceHelpers) by means of the getHelper () method of the Agent class.

The example included in the examples.messaging directory shows in details how to send signed and/or encrypted messages. More in details the SecureSenderAgent and SecureReceiverAgent show the sending part and the receiving part respectively and should be used together. In order to start them just launch a JADE main container with the security service, signature service and encryption service and specifying the SingleUser authentication type as below.

```
java –cp <jade-classes> ;< jade-s-classes> ;< example-classes> jade.Boot
-gui
-services
jade.core.security.SecurityService;jade.core.security.signature.Signature
Service;
jade.core.security.encryption.EncryptionService
-jade_security_authentication_loginmodule SingleUser
s:messaging.SecureSenderAgent
r:messaging.SecureReceiverAgent
```

4.6.13 JADE Test Suite Addon:

The JADE Test Suite is a framework build on top of JADE that permits to create tests that can be executed in a **uniform** and **automatic** way that is:

• **Uniform**: All tests follow the same pattern of execution and produce results in a similar format.

• **Automatic**: All tests do not require user intervention neither during execution or to detect whether the test passed or failed. "***functionality-test***" is the set of tests related to all the aspects of a given functionality. Tests on specific aspects, on the other hand, will be referred to as "***atomic-test***".

4.6.14 JADE Test Suite Addon Example

I have the code at: G:\JADE\add-ons\testSuite. This folder has the build.xml for the ANT and testerList.xml to provide the Tests that need to be conducted. From the Command prompt at the G:\JADE\add-ons\testSuite location type the **ant run** command. The ANT uses the build.xml file and creates the necessary code and runs the JADE and JADE Test Suite. From the JADE Test Suite click on the RUN ALL Toolbar item. This invokes the Test mode and the tests provided in the testerList.xml file are RUN (Figure 4.40).

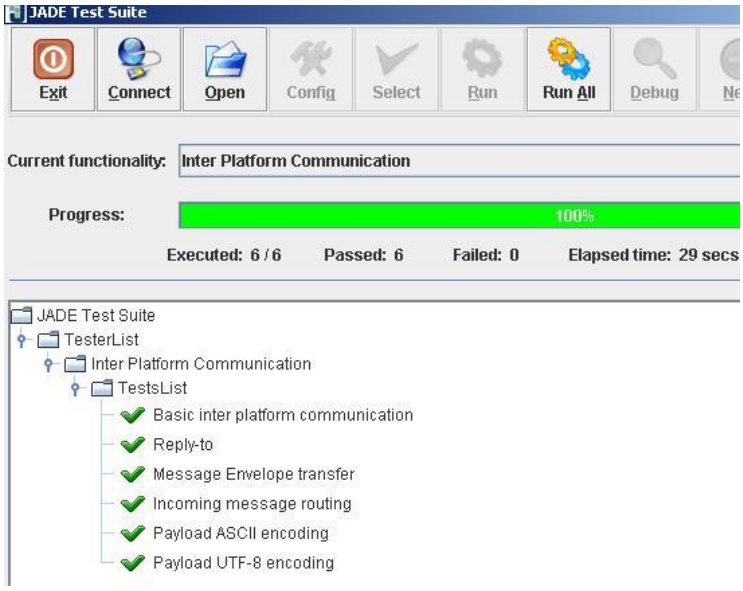


Figure 4.40 JADE Test Suite execution

Figure 4.41 Test Agents deployed in JADE

As shown in Figure 4.41, the Tester Agent is invoked on the JADE platforms and the Test parameters shown earlier are conducted. Thus the Testing occurs in JADE Test Suite.

4.7 Evaluation method: Case studies, real experiences

Table 4.9 Case studies

Sl.No	Context of the case study	Associated content of case study
1	Natural disaster management	Multi-agent systems have a potential to collaborate with each other using their language but the challenge is to make them work intelligently during the situation of catastrophic disaster. In such situations, it is extremely viable to diagnose and dispose resources like ambulances, volunteers, etc. timely, in order to help out people and reduce casualties
2	Resource Management in a Telecom	Like most telecommunication companies and service providers, France Telecom deals with highly heterogeneous applications, ranging from batch oriented to rich interactive multimedia services. Deploying those applications on an utility/grid infrastructure is a promising way to cut operational costs while improving performance (load limitations, QoS, etc.). However, most grid middleware technologies do not deal with interactive (i.e. non-batch) applications. Thus, we propose a grid service platform that dynamically provisions resources for both interactive and batch applications to meet their QoS constraints while ensuring good resource mutualization. Moreover, we believe that relying on an agent-based approach for resource management allows a more flexible, robust and scalable solution
3	Trading agent environment	We design TAGA as a general framework for running agent-based market simulations and games. Our first use of TAGA has been to build a travel competition along the lines that used in the last three year‘s TACs

4.7.1 Details of Case studies assisted evaluation:

Case study 1:

A Framework of Multi-Agent Systems Interaction during Natural Disaster (FMSIND)

Abstract: Multi-agent systems have a potential to collaborate with each other using their language but the challenge is to make them work intelligently during the situation of catastrophic disaster. In such situations, it is extremely viable to diagnose and dispose resources like ambulances, volunteers, etc. timely, in order to help out people and reduce casualties. (Pervez, 2010)

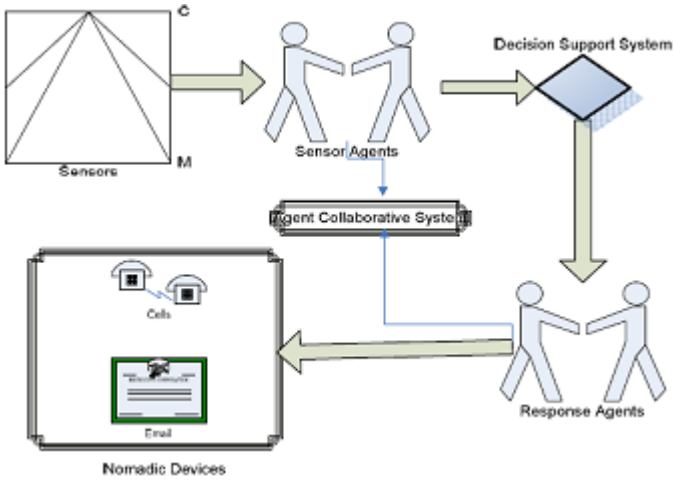


Figure 4.42 Architecture of FMSIND

The key components of our framework are the following (see Figure 4.42).

- **Sensor System** senses the situation (e.g., seismic reading of earthquake or the smoke sensors etc.) and if it is crossing the threshold then it intimates to the Sensor Agents. The sensor devices are used for this purpose e.g., for earthquake detection; the Seismic Reading device can be used, and for fire detection; the usual Fire Alarms can be used.

- **Sensor Agents** sense the situation on-site. These agents have link with sensor devices that give them activation signals, if threshold limit crosses. The sensor device is responsible for activation of sensor agent. Once the sensor reading crosses the threshold, at the same time, it produces the alarm along with the signals to the sensor system.

- **Interactive MAS** consists of agent communication framework like Java Agent Development Framework (JADE)

- **Decision support system** includes learning agents, neural networks, and the components of data warehousing to find the pattern.

- **Response agents** are activated as per instruction of the decision support system

- **Nomadic Devices** like personal digital assistants, email systems, mobile phones are used for intimations and warnings for taking actions.

Our main focus in this framework is on the interaction between the agents at disaster site and resource site (like hospitals, fire-brigade offices, rescue services, etc.). The messages between the agents are sent using agent communication language (ACL). Research in the project of ALADDIN has developed such an algorithm, and it offers several orders-of-magnitude improvement on current state of the art. We use this algorithm in the coordination part of onsite and resource-agents.

Autonomous Learning Agents for Decentralized Data and Information Networks (ALADDIN) is a multi-disciplinary project that deals with the dynamic nature of uncertain distributed and decentralized Intelligent Agent Systems. The project deals with the communication and interaction between the multiple agents to achieve the individual and collective goal. It concerns with three main tasks

- How the interaction between agents can be structured?
- How the class of methods and agents can be used to coordinate for solving the problem during the operations?
- How interactions between these agents can be modelled and simulated?

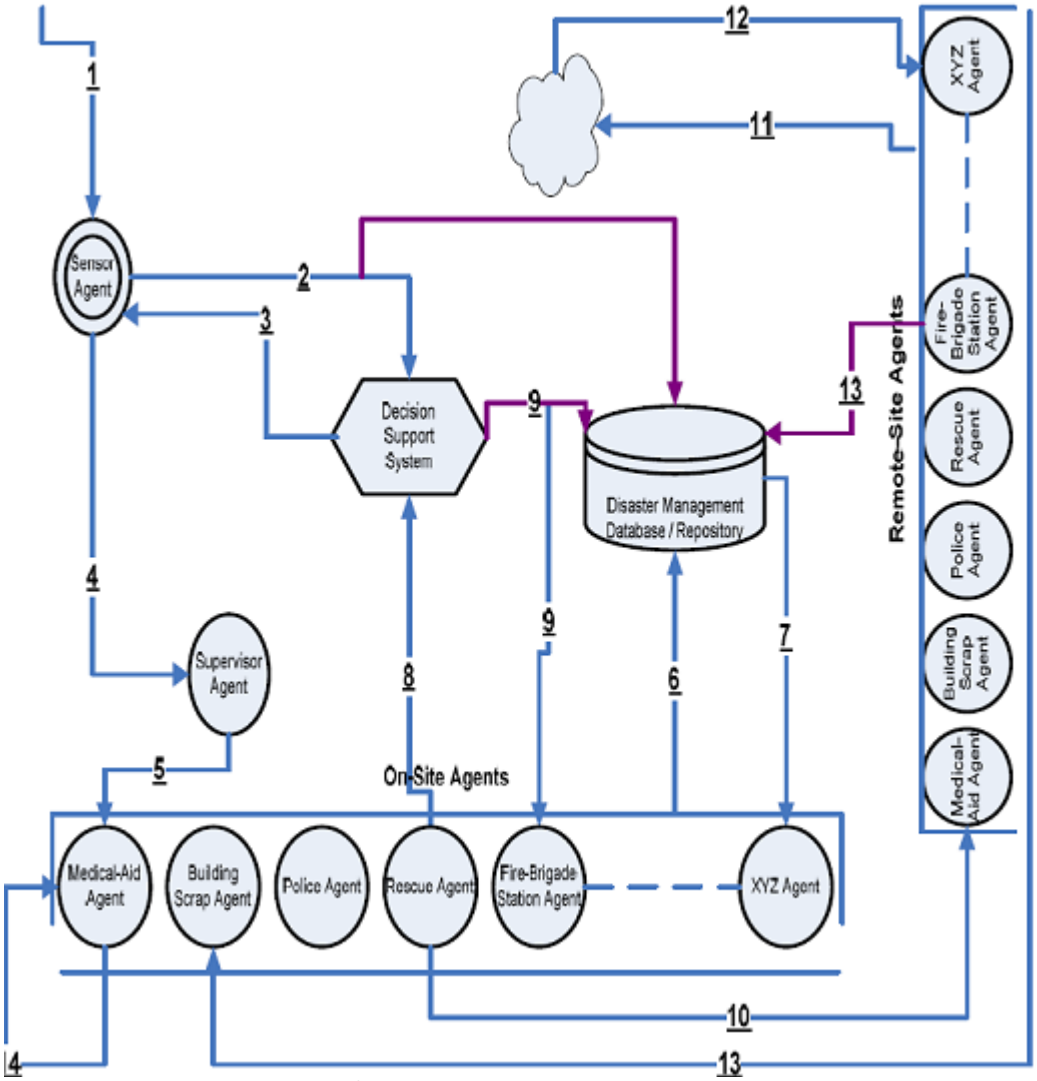


Figure 4.43 Detailed architecture of FMSIND

Sensor Agent (SA) has the responsibility of activating the suspended agents as well as getting information from the sensor system about the loss, and providing this information to the database and decision support system. It also has the responsibility to get data from the decision support system about the on-site required resources.

Supervisor agent has responsibility to get the required resources data from the sensor agent and assign tasks to the on-site coordinating agents. For example, if there is a need of 5 ambulances, the supervisor agent passes the information that the site requires 5 ambulances and assigns duty to the onsite agent to coordinate with the remote-site agents to get this done.

On-site agents get the resources required on on-site from supervisor agents and check their availability by interacting with the disaster database management system. Then, they provide this data to the decision support system to get the plan. They also provide the information about the resources required on on-site to the remote-site agents.

Remote-site agents send the required resource information to the resource handlers as per the addresses provided by the on-site agents. The resource handlers send the reply to the remote-site agents. The remote-site agents then compile the information and send back to the on-site agents.

Decision Support System (DSS) has the responsibility to decide about the number of resources required by the site as per the information provided by the sensor agent. We use the decision support system for devising a plan to assign resources to the on-site agents.

Disaster Management Database System / Repository (DB) has the data about disasters, allocated and deployed resources. It also has the data about the resources availability and their addresses. The resource coordinators have rights to update this information in the database.

Case study 2:

Toward Agent-based Cooperative Resource Management in a Telecom Operator Grid

Abstract: Like most telecommunication companies and service providers, France Telecom deals with highly heterogeneous applications, ranging from batch oriented to rich interactive multimedia services. Deploying those applications on an utility/grid infrastructure is a promising way to cut operational costs while improving performance (load limitations, QoS, etc.). However, most grid middleware technologies do not deal with interactive (i.e. non-batch) applications. Thus, we propose a grid service platform that dynamically provisions resources for both interactive and batch applications to meet their QoS constraints while ensuring good resource utilization. Moreover, we believe that relying on an agent-based approach for resource management allows a more flexible, robust and scalable solution. (Lenica)

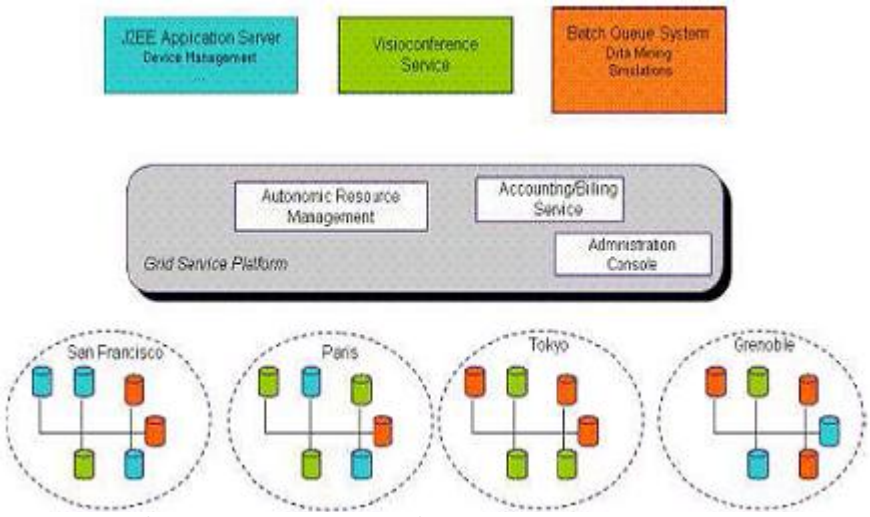


Figure 4.44 Overview of an enterprise grid platform

Architecture for dynamic resources provisioning:

Ensuring good resource management starts by ensuring a good dependability of the resource allocator, so that resources remain available when faults occur. Hence this needs a distributed set of cooperating agents for resource management. Using cooperative distributed agents also helps with scalability and reactiveness issues in such potentially large scale distributed environments. Dealing with heterogeneous applications also implies heterogeneous QoS requirements. To arbitrate amongst the requests, we use **service differentiation**: services are weighted according to a given utility function (for example, economic utility) and resource allocation is prioritized.

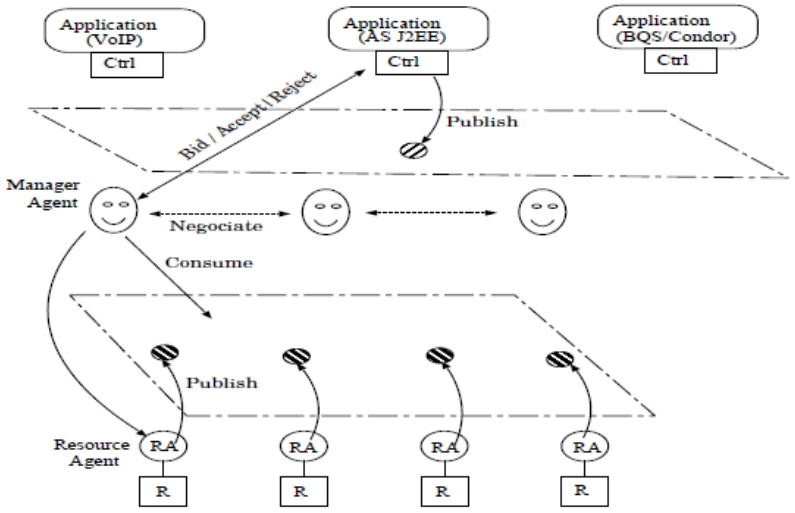


Figure 4.45 A cooperative resource manager

A generic architecture for resource management

Our architecture deals with **resource-centered decision-making**. To achieve this purpose, we propose an agent-based middleware supporting arbitrary resource allocation semantics. The figure 4.45 gives an overview of the general agent-based architecture. It consists of three layers: (i) an **applicative layer** with, for each application, a controller responsible for maintaining all metadata associated to the application, monitoring its performance and interfacing application and grid middleware; (ii) the **core layer** of resource management agents, responsible for resource dissemination, resource discovery and scheduling; (iii) an **infrastructure layer** consisting of virtualized grid resources and resource agents.

Each resource is handled by a simple agent (called resource agent). Its role is to put in relation for registration purpose the associated resource with a manager agent. To that extent, a shared-space based coordination model is used. A high-level description of the registration process can be stated as follows.

Upon activation, the resource agent deposits a request for registration in the shared memory space. It is consumed by the first available manager agent. This agent takes in charge the concerned resource (distinct resources can be managed by the same agent). To enhance resource management dependability, a simple replication schema is applied to managers: each resource possesses a single primary manager (for consistency sake) but knows also of secondary managers or **co-managers**. The designation of the co-managers falls to the manager (they may choose neighbours or clone themselves). Once the co-manager list determined, the manager transmits all the references (both of manager and co-managers) to the resource agent.

In turn, the resource agent is responsible for detecting a manager failure and electing a new manager among the co-managers. From an implementation point of view, the choice of a shared memory architecture is affected by scalability issues. As global memory is hardly achievable, we will consider solutions like distributed tuplespaces (e.g. Comet).

Similarly the resource allocation process uses a shared memory space to enable application controllers to request resources from the manager agents. The allocation is done through a **Contract-Net** type protocol. The Contract-Net protocol (CNP) and its extensions is a widely used negotiation protocol within the field of multi-agent systems. Primarily designed for task allocation, it is also perfectly suited to multi agent resource allocation. It consists of four interaction phases, involving two roles: manager and bidder. The following paragraph describes the first two phases: the announcement phase where the agent initiating the negotiation (i.e. the manager) makes a call for proposal to a number of partners (the bidders), and the bidding phase, where the bidders send their proposals to the manager. The last phases deal with assignment and confirmation. Following the CNP terminology, the application controller stands as the **manager** and the allocation agents as the **bidders**.

The allocation protocol is further detailed below.

On the manager side: The **call for proposal** (CFP) mentions resource profiles (hardware or software), the application priority and a **deadline**. The CFP is released through the shared memory space. Upon reception of the proposals, the best suited is selected, based upon application functional and QoS constraints. Contrariwise to a classic CNP, the controller does not wait for the CFP deadline expiration to start evaluating the proposals. If it has received a satisfactory proposal, it can still wait for a better one until the expiration of the CFP deadline (without jeopardizing the anterior proposal). Hence, the choice of a proposal lies on a reactivity / optimality tradeoff.

On the bidder side: The CFPs are addressed in order of priority, so as to enforce service differentiation. Agents compete to bid for a CFP. A bid mentions the resources offered, possible QoS metrics and a deadline. The agents may submit partial bids and perform multiple bids simultaneously. The consequences of this statement are multiple. On the first hand, partial bids have to be processed on the manager-side. Aggregation of partial bids may be a complex application dependent task. To avoid having to re-develop ad-hoc controllers for each application, we choose to keep controllers as simple and generic as possible. On the counterpart, the resource management middleware is endowed with the task of elaborating satisfying proposals, by means of agent cooperation if necessary. On the second hand, considering there are several sources of tasks (the application controllers) and that the agents have only limited resources to actually execute the tasks, we face what Schillo et al. in call the **Eager Bidder Problem**: the agents have to decide "in how many of the concurrent negotiations they intend to participate and how they should handle their commitments with respect to the local resources at hand". At the current stage of our model and for simplicity's sake, we state that agent commitment is definitive (no disengagement allowed until deadline expiration).

Case study 3:

A case study in the TAGA trading agent environment:

We design TAGA as a general framework for running agent-based market simulations and games. In the competition, ***customers travel*** from City A to City B and spend several days before flying back. A ***travel package*** includes a round-trip flight ticket, corresponding hotel accommodation and tickets to entertainment events. A ***travel agent*** competes with other travel agents in making contracts with customers and purchasing the limited travel services from the ***Travel Service Agents***. Customer selects the travel agent with best travel itinerary. (Zou).

TAGA provides a flexible framework to run the travel market game. Figure 4.46 show the structure of TAGA. The collaboration and competition among **six types of agents** who play different market roles simulate the real world travel market.

Figure 4.46 TAGA Architecture

The ***Auction Service Agent*** **(ASA)** operates all of the auctions in TAGA. Supported auction types include English and Dutch auctions as well as other dynamic markets similar to Priceline.com and Hotwire.com.

A ***Service*** **Agent (SA)** offers travel related service units such as airline tickets, lodging and entertainment tickets. Each class of travel related service has multiple providers with different service quality level and with limited service units. It allows other agents to query its description (e.g. service type, service quality, location) and its inventory (the availability or price of a certain type of service unit). Other agents may directly buy the service units through published service interface. SA also bids intentionally in the auctions to sell its good, e.g. listing its goods in auction and wait for the proper buyer.

A ***Travel Agent*** **(TA)** is a business that helps customers acquire travel service units and organize travel plan. The units can be bought either directly from the service agents, or through an auction server.

A ***Bulletin Board Agent*** **(BBA)** provides a mechanism helping customer agents find and engage one or more travel agents.

A ***Customer Agent*** **(CA)** represents an individual customer who has particular travel constraints and preferences. Its goal is to engage one or more TAs, negotiate with them over travel packages, and select one TA that is able to acquire all needed travel service units.

The ***Market Oversight Agent (MOA)*** monitors the game and updates the financial model after each reported transaction and finally announces the winning TA when the game is over.

The basic cycle of the TAGA game has the following five stages:

1. A customer-generating agent creates a new customer with particular travel constraints and preferences chosen from a certain distribution.

2. The CA sends the customer's travel constraints and preferences to the BBA in the form of a CFP (call for proposal) message. The BBA forwards the CA's CFP message to each of the TAs that has registered with it. Each TA considers the CA's CFP independently and decides whether and how to respond.

3. When deciding to propose a travel package, The TA contacts the necessary ASAs and SAs and assembles a travel itinerary. Note that the TA is free to implement a complex strategy using both aggregate markets (ASAs) as well as direct negotiation with SAs. The proposal to the CA includes the travel itinerary, a set of travel units, the total price and the penalty to be suffered by the TA if it is fail to complete the transaction.

4. The CA negotiates with the TAs ultimately selecting one from which to purchase an itinerary based on its constraints, preferences and purchasing strategy (which might, for example, depend on a TA's reputation).

5. Once the TA has a commitment from the CA, it attempts to purchase the units in the itinerary from the ASAs and SAs.

There are two possible outcomes: the TA acquires the units and **completes the transaction** resulting in a satisfied CA and a profit or loss for the TA, or the TA is unable or unwilling to purchase all of the units, resulting in an **aborted transaction** and the invocation of the penalty (which can involve both a monetary and a reputation component).

Chapter 5

Results and Discussions

The actual outputs from the Theory and the Practical analysis conducted in preceding section is tabulated in form of reports, graphs and discussed for their significance and value created for the end user.

5.1 Results: Evaluation methods - Theory assisted

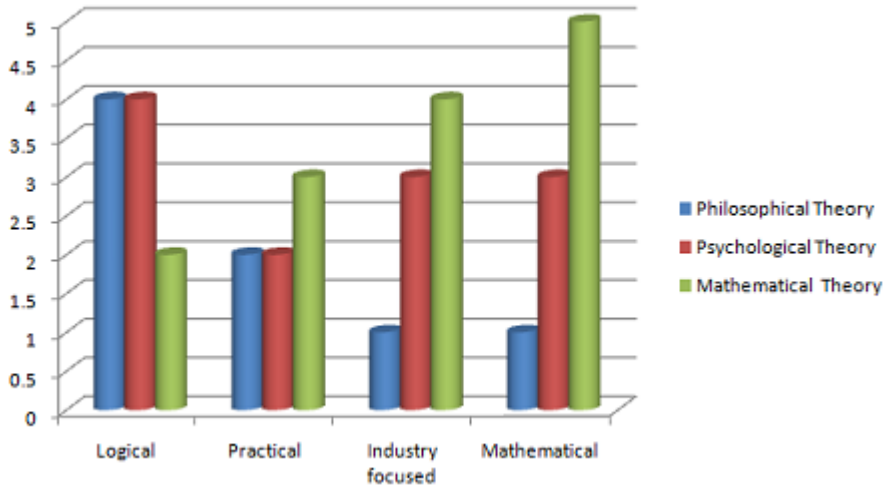


Figure 5.1 Results for Theories

My observations and grading of the three theories for my study using different metrics are shown in Figure 5.1

Considering the **Logic**, Philosophical and Psychological theories provide better vision and direction based on the concepts and theories of AI. They capture the essence of the mindset and provide the high level feature of Agents and how they affect the society.

Considering the **practicalities**, Mathematical Theory provides analysis based on imperial analysis and provides a frame work for capturing the essence of agent theory in a Mathematical model and see how the theory can be made practical enough for project implementation.

Considering the **Industry** focus, Mathematical model provides standard framework for projects to be implemented in a factory subject to realization of the mathematical model under certain assumptions that make the Industry to produce products that meet the public expectations. Other theories provide a generic view and suggestions for approaching the market with the products.

Considering **Mathematical** aspect, the Mathematical theory provides well defined framework to work with and provide the base for providing the means to design and develop the project artefacts related to multi agent system.

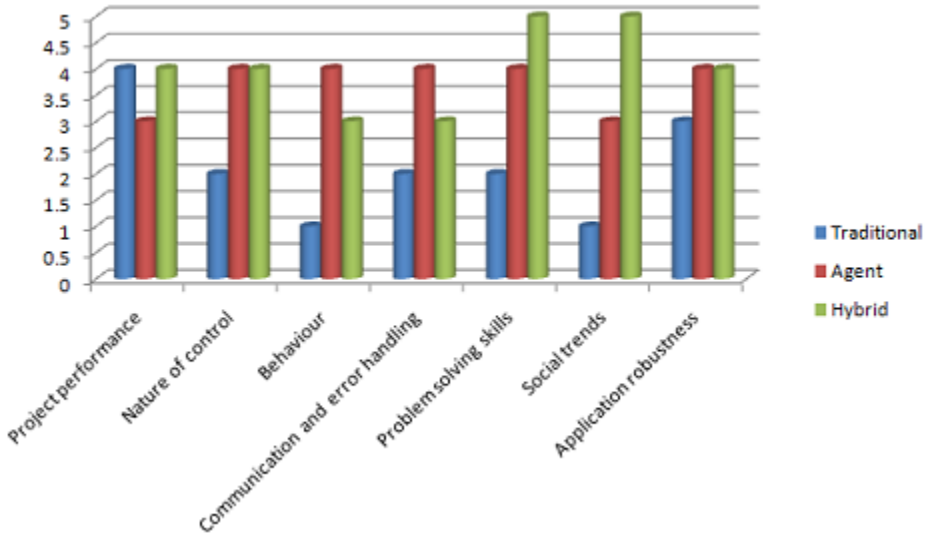


Figure 5.2 Comparisons of Theories

Discussion: Figure 5.2 shows another set of features that are used to compare the Traditional versus Agent based projects and if the better of these two are used how the Hybrid project could perform.

Project performance is good when accomplished using Traditional process and Tools based on Object oriented technologies. Since the Industry has matured, developers will find it easy to work on this as they have enough documentation, experts and quick resolutions. On the other hand Agent specific projects lack maturity. But their actual performance is better and more realistic and closer to human thinking and actions. A combination of these types makes Hybrid projects better but due to the issues in integration the performance will not be very much more than expected.

Nature of control is great for Agent as it is systematically distributed and no tight coupling like Remote function calls or more traffic due to Object based design. Agents provide behaviours and autonomous features that make them similar in nature to humans and hence cross chats are avoided as on an object based platform.

Behaviour feature can easily be generated in Agents but it needs complex attributes in Objects. For creating different types of moods like humans Agent programming makes it easy to code and execute, but for Objects more data and wrapper classes need to derive the same features in Object code. Moreover Agent based IDE have templates and code generators that can make the implementation easy and fast. However the Object technologies do not have such default features.

Communication and error handling: In Objects the Communications are chattier and the invocation is made by external calls, hence objects are mainly passive in nature. However Agents have self decision capabilities and hence there is no need of second call or talk to other agent. Hence they are mainly active and need less talk. Due to the above design error handling also needs less vigilance. However the nature of capturing errors and propagating in Agents is complex and needs better management and synchronisation.

Problem solving skills is better in agent technology as compared to Object technology. Agents by design have features to work as self entities distributed objects that can evenly and independently work for long time to resolve the sub programs and return results without much chatting or referencing outside entities. Also there are other design concepts agents have that Objects don't have and need more complexities to do similarly.

Social trends concept is more in Agents as compared to Objects. In fact by default Objects do not exhibit Social behaviour and cannot be easily compared to agents in this aspect.

Application robustness can be compared to both technologies and they match in one way or other. There are not much differences and each one has its merits and demerits. So the robustness can be of similar nature. Objects have good IDE and matured debugging features than Agent technology but Agents are better post implementation and are better off for reduced maintenance and foot prints can be adjusted to the needs of the client.

5.2 Results Evaluation methods : Practical

Figure 5.3 shows the details of the comparison for the 15 criteria for the various tools.

Criteria	Tools	*JADE*	*DECAF*	*AgentBuilder*	*Zeus*	*JAFMAS/JIVE*	*Jack*	*AgentTool*	*Madkit*
Methodology (1)		0	0	4	4	3	0	3	3
Learning facility (2)		0	3	1	1	1	0	3	2
Step transition (3)		0	0	3	2	2	0	3	2
Tool flexibility (4)		3	0	1	1	2	3	0	3
Communication (5)		4	2	4	4	2	3	2	3
Debug utility(6)		4	2	4	4	1	2	2	3
Development support (7)		0	2	4	4	2	1	4	0
Implementation support (7)		0	0	4	4	2	1	2	0
MAS management (8)		4	0	3	3	0	0	1	4
Implementation simplicity (9)		2	3	2	2	1	2	3	1
Database (10)		0	0	1	2	0	3	0	0
Code generation (11)		0	0	1	3	1	0	1	0
Code extensibility (12)		4	1	1	2	1	4	0	3
Deployment (13)		4	1	2	2	1	2	1	3
Documentation (14)		4	1	4	4	1	3	1	3
Total /60		**29**	**15**	**39**	**42**	**20**	**22**	**26**	**30**

Figure 5.3 General Results Grid

Zeus and AgentBuilder obtained the two best scores (42 and 39, respectively) of the eight evaluated tools. These tools have an abundant documentation. They offer several graphic interfaces helping the development. Moreover, they have a debug utility. Each one proposes a development methodology. However, these two tools are very complex and they require significant training to be understood reasonably well, and much more to be mastered.

AgentBuilder requires knowledge about RADL (Reticular Agent Definition Language), ontology management, the execution engine, the protocols used and the mastering of other interfaces.

Zeus also requires a good bit of training, notably to master the “role modeling” technique, as it is essential in Zeus. It is also necessary to understand the various editors that provide several more or less advanced services (ontology, tasks, resources, rules, interaction protocols, system visualization, etc.). Both Zeus and AgentBuilder have weaknesses at the extensibility and flexibility levels

I have personally used JADE. It is USED MOSTLY IN all the major project and adheres to FIPA, an IEEE approved standard.

Another set of comparisons as related to the SDLC of the Project development is shown in Figure 5.4

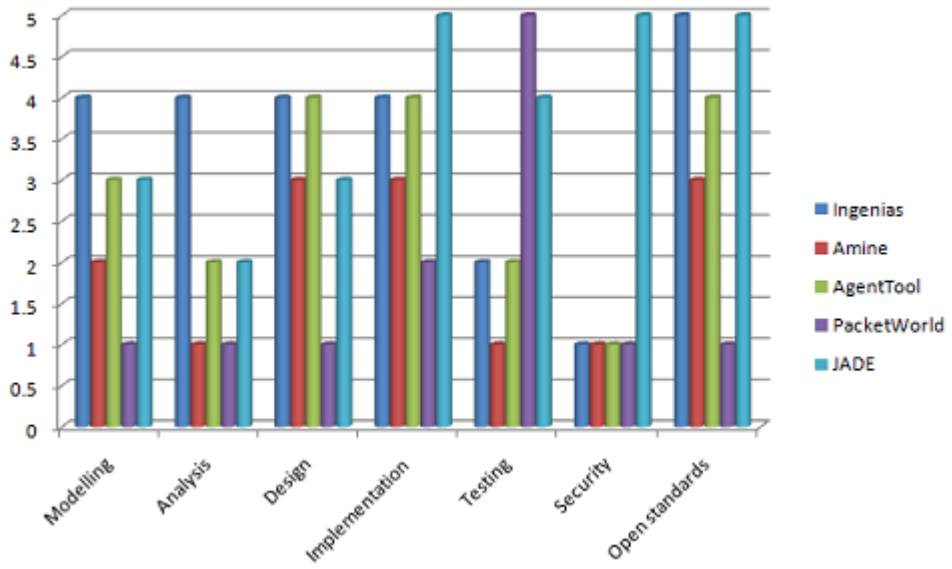


Figure 5.4 Results for Practical

Modelling: MAS modelling in INGENIAS is facilitated by a graphical editor, automatic code generation and validation tools. It uses meta-models and hence the work is systematic. Amineplatform uses Ontology based Model generation and hence has less GUI interfaces than INGENIAS. agentTool realises partial Agent classes and uses pre constructed UML from other tool. Hence it is not a good modelling tool. Packet world is a test environment hence not a modelling tool and GUI is used for configuration. JADE can model Agents but have restricted editors for Ontology / XML manipulations only.

Analysis: In INGENIAS analysis is inbuilt to the models. There is no separate editor exclusive for analysis. Hence it is an average way to do analysis in INGENIAS editor. Amine is not an Analysis tool and is mainly for Design. Hence not a great source for Analysis. Agent Tool is not yet matured for Analysis, but it is based on MaSE which has great Analysis features. Packet world is a test environment hence not an analysis tool and GUI is used for configuration. JADE has limited Analysis features and is mainly used as middleware and deploying of Agents.

Design: INGENIAS has Design features similar to ZEUS or Agent Tool and hence is a good design tool. Amine has dedicated tools using various entities like Ontology, Prolog, XML Structures and backed by the Conceptual Graph theory concept. Hence it is a good analysis tool. Agent Tool is not matured for Design, as it is based on MaSE which has great Design features. Packet world is a test environment hence limited tool for Agent Config design. JADE is good for design of Agents and Society for supporting Multi agent communication.

Implementation: INGENIAS has features that can generate java code for JADE platform. Amine has dedicated tools using various open standards and programming language like Prolog. Hence it is a good implementation tool. Agent Tool is matured for Implementation, as it is based on MaSE. Packet world is a test environment hence limited tool for implementation. JADE is good for design of Agents and Society for supporting Multi agent communication, hence well for implementation.

Testing: Packet world and JADE combined with JUnit form a good testing tool.

Security: JADE provides the best security at code and config level and hence tops the list. Others do not have elaborate features and has to be added separately to code.

Open standards: All of the products do have certain level of compliance and use Java as the main platform. Packet world is restricted to its own standard and not much flexible.

5.3 Results for case studies

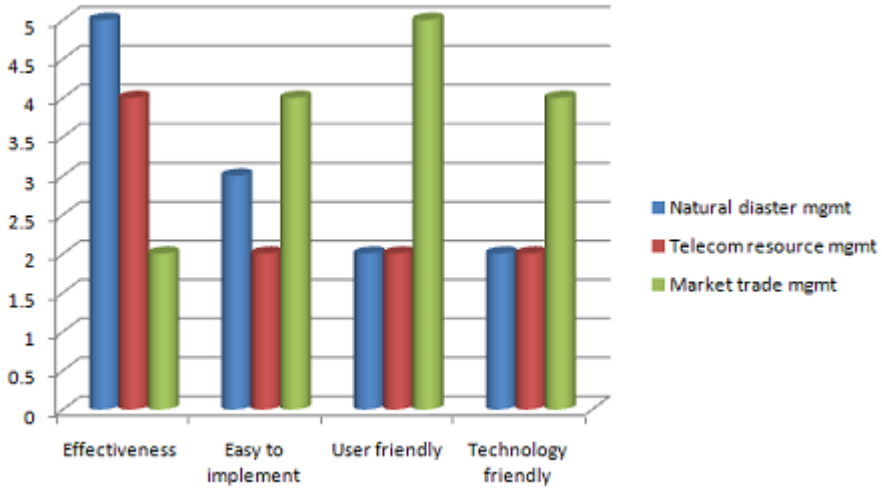


Figure 5.5 Results for Case studies

Effectiveness of the Solution to the Problem: Case study one related to Natural disaster management provides a good solution to the problem in hand. It uses various sensors to capture events and logs that it a database where AI based algorithms generate suitable notifications. Case study two related to Telecom resource management is quiet effective and is a new way to deal resource crunch. Case study three related to Market trade management is also effective in dealing the situation of allocation of Market resources. However this concept has been addressed earlier and hence not a unique and new innovative idea. Hence score is less.

Easy to implement: Natural disaster management has an average difficulty in implementing, whereas Telecom resource management is difficult to design and implement. It needs complex handling of resources. Market trade management is an average complexity to implement and it needs more process to manage and handle the huge transactions based on business volumes.

User friendly: Natural Disaster Mgmt is not so User friendly as it needs Technical experts working on various sensors and integrates the Hardware and Software interfaces. The GUI might look good but the complexity of the background processes is not easy to manage.

Telecom Resource Mgmt is also not so user friendly as it has lot of background and hidden interfaces away from the end users; Market Trade Mgmt is user friendly as it has to be because the real humans need more interactions for completing transactions.

Technology friendly: Market trade management is technology friendly and does not need complex Technologies to implement. However the other case studies have complex needs to compute, manage and repair.

Chapter 6

Conclusions and Recommendations

In conclusion this Dissertation has provided a brief introduction to the Multi Agent system project foundations, along with the development tools, methodologies, process and testing features. Project team has been exposed to the know-how of the Agent technology and how the theory can be applied to a practice and demonstrated using relevant case studies.

The main objectives of dissecting the MAS and deriving the true value of project have been accomplished using the most suitable theories and tools. The details provided in this Dissertation should provide the necessary knowledge and facts to take up the Agent technology into real world projects and generate the necessary products to enhance the social values and generate good returns to the utilities developed.

The future of the Agent technologies is bright and the world is moving from the Real world to the Virtual World by harnessing the potential of Agent technology and incorporating the merits and innovations of the AI Industry backed by the research conducted by many Institutes of National importance.

I would like to recommend a simple layout of the project package details of which are highlighted in following section.

6.1 Global solution to MAS projects

Provide a common easy to use and not any technology agnostic solution for IT project team to follow, a life cycle model as a global solution to the common problems in MAS project implementation

Project details

A high level MAS Package Structure with sub packages (like Architecture) and Modules (like Agents) is depicted in Figure 6.1 It provides a bird's view of the MAS system that is proposed to be developed to demonstrate the project values.

The package is a general high level view to show what all constitutes the MAS solution. This is a bird's view of the overall package and this need to be detailed and customised as per the needs of the individual projects.

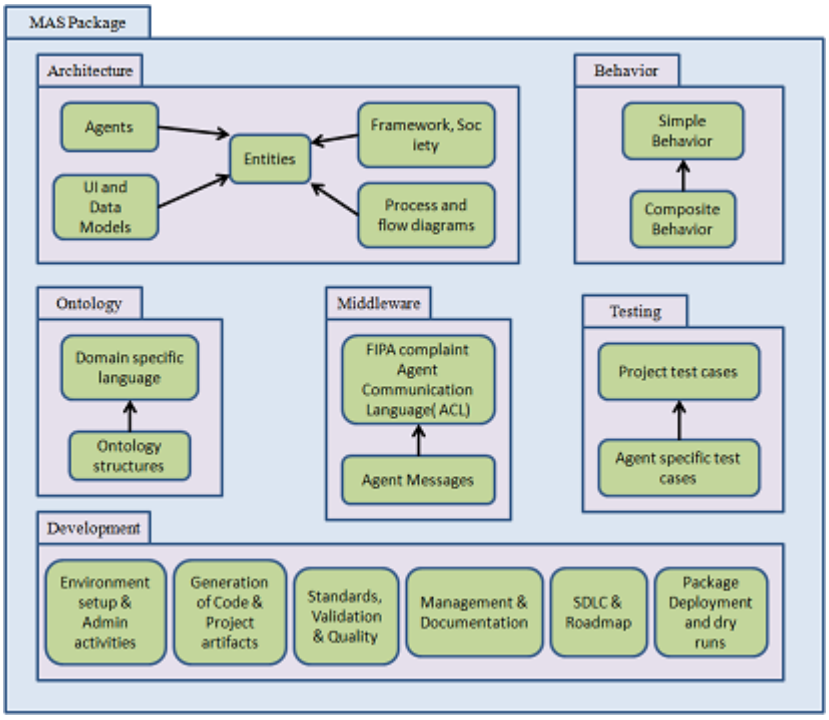


Figure 6.1 MAS Package Structure

Appendix I

IDK Code for Hello World

```
package ingenias.jade.agents;

import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.*;
import ingenias.jade.*;
import ingenias.jade.smachines.*;

import java.io.IOException;
import java.io.InterruptedIOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.lang.reflect.InvocationTargetException;
import java.net.InetAddress;

import jade.core.*;
import jade.core.behaviours.*;

import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.domain.DFService;
import jade.domain.FIPANames;
import jade.lang.acl.ACLMessage;
import ingenias.jade.*;
import java.util.*;
import ingenias.jade.exception.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import ingenias.editor.entities.RuntimeFact;
import ingenias.jade.components.*;
import ingenias.jade.mental.*;
import ingenias.editor.entities.ApplicationEventSlots;
import ingenias.editor.entities.Interaction;
import ingenias.editor.entities.MentalEntity;
import ingenias.editor.entities.ObjectSlot;
import ingenias.editor.entities.RuntimeEvent;
import ingenias.editor.entities.RuntimeFact;
import ingenias.editor.entities.RuntimeConversation;
import ingenias.editor.entities.Slot;

import ingenias.jade.components.Task;
import ingenias.jade.graphics.*;
import ingenias.jade.MentalStateManager;
import ingenias.exception.InvalidEntity;

public class HellowWorldAgentJADEAgent
              extends JADEAgent {

              public HellowWorldAgentJADEAgent(){
              super(new HellowWorldAgentProtocol(),new
HellowWorldAgentInteractionLocks());
              }
```

```
private boolean initialiseNonConversationalTask(Task tobject) {
          boolean initialised=false;
          Vector<String> repeatedOutputs=new Vector<String>();
           Vector<String> nonExistingInputs=new Vector<String>();

           if (tobject.getType().equals("DeleteNonUsedEntitiesTask")){
            Vector<MentalEntity> expectedInput=null;
            RuntimeFact expectedOutput=null;
            RuntimeEvent expectedOutputEvent=null;
                    RuntimeConversation expectedInt=null;
            ingenias.jade.components.Resource expectedResource=null;
                    ingenias.jade.components.Application expectedApp=null;
                    boolean allEntitiesExist=false;
                    TaskOutput to=null;
                    to=new TaskOutput("default");


             if (this.getLM().canBeDeleted("NONSENSEENTITY")){
             expectedInput=this.getMSM().getMentalEntityByType("NONSENSEENTITY");
             if (expectedInput.size()==0){
                          nonExistingInputs.add("NONSENSEENTITY");
                      } else {
                         addExpectedInputs(tobject,
"NONSENSEENTITY","1",expectedInput);
              addConsumedInput(to,"NONSENSEENTITY",expectedInput);
                      }
             allEntitiesExist=allEntitiesExist || expectedInput.size()!=0;
              }

               tobject.addOutput(to);

             initialised= allEntitiesExist;

                if (!allEntitiesExist){
                     StringBuffer nonexisting=new StringBuffer();
                     for (int j=0;j<nonExistingInputs.size();j++){

       nonexisting.append(nonExistingInputs.elementAt(j).toString()+",");
                     }
                }
                return initialised;
           }


           nonExistingInputs.clear();
                repeatedOutputs.clear();
          if (tobject.getType().equals("Greet")){
             Vector<MentalEntity> expectedInput=null;
             RuntimeFact expectedOutput=null;
             RuntimeEvent expectedOutputEvent=null;
                     RuntimeConversation expectedInt=null;
             ingenias.jade.components.Resource expectedResource=null;
                     ingenias.jade.components.Application expectedApp=null;
                     boolean allEntitiesExist=true;
                     TaskOutput to=null;
                     to=new TaskOutput("default");




expectedInput=this.getMSM().getMentalEntityByType("GreetingsContainment");
             if (expectedInput.size()==0 && !("1".equals("0..n"))){
                          nonExistingInputs.add("GreetingsContainment");
                      } else {
                         addExpectedInputs(tobject,
"GreetingsContainment","1",expectedInput);
              addConsumedInput(to,"1",expectedInput);
                      }
             allEntitiesExist=allEntitiesExist&& expectedInput.size()!=0;
```

```
        // Default application for all tasks executed within a conversation

expectedApp=(ingenias.jade.components.Application)getAM().getApplication("YellowP
ages");
             tobject.addApplication("YellowPages",expectedApp);

             /**/

             tobject.addOutput(to);
             initialised= allEntitiesExist;

              if (!allEntitiesExist){
                    StringBuffer nonexisting=new StringBuffer();
                    for (int j=0;j<nonExistingInputs.size();j++){

      nonexisting.append(nonExistingInputs.elementAt(j).toString()+",");
                    }

                    MainInteractionManager.log("Non conversational initialisation
discarded task "+tobject.getType()+
                          " to achieve goal Greet_user because did not have all
preconditions. Missing elements are:"+nonexisting+". Currently, the MS has the
following elements:"+this.getMSM().getAllMentalEntities(),getLocalName()+"-
"+tobject.getType());
              }
              return initialised;
             }


             return false;
      }

      private boolean initialiseConversationalTask(RuntimeConversation
conversation, Task tobject) {
         boolean initialised=false;
         Vector<String> nonExistingInputs=new Vector<String>();
         Vector<String> repeatedOutputs=new Vector<String>();
         boolean validConversationType=false;

         if (tobject.getType().equals("DeleteNonUsedEntitiesTask")){
           Vector<MentalEntity> expectedInput=null;
           RuntimeFact expectedOutput=null;
           RuntimeEvent expectedOutputEvent=null;
                   RuntimeConversation expectedInt=null;
           ingenias.jade.components.Resource expectedResource=null;
                   ingenias.jade.components.Application expectedApp=null;
                   boolean allEntitiesExist=false;
                   TaskOutput to=null;
                   to=new TaskOutput("default");
                   tobject.setConversationContext(conversation);


              if (this.getLM().canBeDeleted("NONSENSEENTITY")){

expectedInput=this.getMSM().obtainConversationalMentalEntityByType(conversation,"
NONSENSEENTITY");
             if (expectedInput.size()==0){
                         nonExistingInputs.add("NONSENSEENTITY");
                      } else {
                         addExpectedInputs(tobject,
"NONSENSEENTITY","1",expectedInput);
             addConsumedInput(to,"NONSENSEENTITY",expectedInput);
                      }
             allEntitiesExist=allEntitiesExist|| expectedInput.size()!=0;
              }

               tobject.addOutput(to);
```

```
            initialised= allEntitiesExist;
               if (!allEntitiesExist){
                    StringBuffer nonexisting=new StringBuffer();
                    for (int j=0;j<nonExistingInputs.size();j++){

       nonexisting.append(nonExistingInputs.elementAt(j).toString()+",");
                    }
               }
               return initialised;
           }

            return false;
       }

       // This method returns the tasks this agent can perform in
       // order to satisfy the goal

       public Vector tasksThatSatisfyGoal(String goalname){
         Vector tasks=new Vector();
         Vector<String> typesOfConversation=null;

       //**********************************
         // Non conversational tasks evaluation
         //**********************************

         if (goalname.equals("Greet_user")){

         {
         boolean canbescheduled=false;
              Task tobject=null;
                           // If a conversational initialization fails, a
conventional one is tried
                            tobject=new
GreetTask(ingenias.jade.MentalStateManager.generateMentalEntityID());

canbescheduled=initialiseNonConversationalTask(tobject);
                           if (canbescheduled){
                                 MainInteractionManager.log("Scheduled task
"+tobject.getType()+" to achieve goal Greet_user",getLocalName()+"-
"+tobject.getType());
                                 tasks.add(tobject);
                           }
           }  }

      Task tobject=new
DeleteNonUsedEntitiesTask("DeleteNonUsedEntitiesTask","DeleteNonUsedEntitiesTask"
);
         boolean canbescheduled=initialiseNonConversationalTask(tobject);
              if (canbescheduled && IAFProperties.getGarbageCollectionEnabled()){

                           tasks.add(tobject);
              }
        return tasks;
       }


       /**  Initializes the agent
        */
       public void setup() {
              super.setup();
              Vector<String> ttypes=new Vector<String>();

         ttypes.add("Greet");

         if (IAFProperties.getGraphicsOn())
          this.getGraphics().setKnownTasks(ttypes);
```

```
    // Interactions started by this agent

    boolean continueInit=false;
    // Interactions where this agent acts as collaborator

   // These are the initial goals of the agent. Goals determine
    // which task to execute first
    ingenias.editor.entities.StateGoal sg=null;
    ingenias.editor.entities.RuntimeFact ff=null;
    Slot slot=null;
    ObjectSlot oslot=null;
    ingenias.jade.components.Application app=null;
     sg= new ingenias.editor.entities.StateGoal("Greet_user");
    sg.setState("pending");
       try {
           this.getMSM().addMentalEntity(sg);
    } catch (InvalidEntity e1) {

           e1.printStackTrace();
    }

  ff= new GreetingsContainment();

        try {
                      this.getMSM().addMentalEntity(ff);
              } catch (InvalidEntity e) {

                      e.printStackTrace();
              }
              //Initializing the applications panel in the manager

              Vector events=null;
              RuntimeEvent event=null;

        //Initial applications assigned to the agent
        Vector actions=null;
        Vector evetns=null;

    // Panel creation for interaction control
    // This panel shows a button for each interaction it starts.
    // If this agent does not start any interaction, a label showin
    // a message "DOES NOT START ANY INTERACTION" will appear

    java.awt.event.ActionListener ifPressed=null;

     // Final Graphics initialization
     if (getGraphics()!=null)
      getGraphics().startAgentDebug();

     // To indicate that the MSP can start
     this.agentInitialised();

        }

        /*  Obtains a DFAgentDescription array that describes the different roles
an agent can play,    *@return    Roles played        */
        public DFAgentDescription[ ] getDescription() {
                DFAgentDescription[] result=null;
               Vector playedRoles=new Vector();
                DFAgentDescription dfd=null;
                  dfd = new DFAgentDescription();
                  ServiceDescription sd=null;

                  result=new  DFAgentDescription[playedRoles.size()];
                  playedRoles.toArray(result);
               return result;

        } }
```

Appendix II

Amineplatform MAS Renaldo code

RenaldoApplication.java class

```
package aminePlatform.applications.mas.renaldo;

import aminePlatform.mas.amineJade.AmineJadeMas;
import aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent;
import aminePlatform.guis.util.AmineFrame;
import java.io.*;

/**
 * <p>Title : RenaldoApplication class</p>
 * <p>Description : RenaldoApplication is a simple example that
* illustrates  how to setup a new MAS </p>
 */

public class RenaldoApplication {
  public static String amineLogo =
AmineFrame.class.getResource("AminePlatform.gif").getPath();
  public static String aminePlatformDirPath = new
File(amineLogo).getParentFile().getParentFile().getParentFile().getPath();
  public static String amine2DirPath = new
File(amineLogo).getParentFile().getParentFile().getParentFile().getParentFile().g
etParentFile().getPath();
  public static String renaldoDirPath;
  static {
    StringBuffer absoluteFileName = new StringBuffer(aminePlatformDirPath);
    absoluteFileName.append(System.getProperty("file.separator"));

absoluteFileName.append("samples").append(System.getProperty("file.separator"));
    absoluteFileName.append("mas").append(System.getProperty("file.separator"));

absoluteFileName.append("renaldo").append(System.getProperty("file.separator"));
    renaldoDirPath = absoluteFileName.toString();
  }

  public static void main(String[] arguments) {
/** In a String argument, define parameters a of specified agent
         the structure of this string is :
         agent's name:the path of agent's class(arguments separated by a space)
         John.pcg is a Prolog file that contains the agent's planning program
         ontology.xml is the ontology used by the agent
*/
    String fileJohn = renaldoDirPath + System.getProperty("file.separator") +
"John.pcg";
    String fileArthur = renaldoDirPath + System.getProperty("file.separator") +
"Arthur.pcg";
    String fileEnvironment = renaldoDirPath +
System.getProperty("file.separator") + "Environnement.pcg";
    String fileOntology = renaldoDirPath + System.getProperty("file.separator") +
"ontology.xml";

    String arg1 =
"John:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(" +
        fileJohn + " " + fileEnvironment + " " + fileOntology + ")";

    String arg2 =
"Arthur:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(" +
        fileArthur + " " + fileEnvironment + " " + fileOntology + ")";
```

```
    // Old version
    //String arg1 =
"John:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(John.pcg
Environnement.pcg ontology.xml)";
    //String arg2 =
"Arthur:aminePlatform.agent.amineJade.ppcgAgent.PPCGAmineJadeAgent(Arthur.pcg
Environnement.pcg ontology.xml)";
// put these arguments in an array
    String[] args = {arg1, arg2};
// call the constructor of a MAS by specifying its name and the agent arguments
array
    new AmineJadeMas("Renaldo", args);
// you can retrieve each agent object by calling the static getAgentById method.
The Id is specified as a String
    PPCGAmineJadeAgent John = (PPCGAmineJadeAgent)
AmineJadeMas.getAgentById("John");
// use these agent objects to call the predefined methods to satisfy of send
messages between agents.
// in the following case john tries to satisfy the goal (as a String) expressed
as a CG
    John.satisfyGoal("Animal(John, Goal)::[Animal : John]<-pat-[SatisfyHungry]-
intensityOf->[Hungry = _Level]");
  }
}
```

ENVIRONMENT.PCG (Opens from AminePlatform Prolog+CG editor)

```
Environment( Food ):: [Food = "honey"] -
             -locatedIn-> [Position = [3,11]],
             -exists-> [Quantity = 100 ].
Environment( Food ):: [Food = "honey"] -
             -locatedIn-> [Position = [24,6]],
             -exists-> [Quantity = 250 ].
Environment( Food ):: [Food = "fish"] -
             -locatedIn-> [Position = [20,11]],
             -exists-> [Quantity = 30 ].
Environment( Food ):: [Food = "fish"] -
             -locatedIn-> [Position = [11,1]],
             -exists-> [Quantity = 50 ].
Environment( Food ):: [Food = "cheese"] -
             -locatedIn-> [Position = [23,3]],
             -exists-> [Quantity = 300 ].
Environment( Food ):: [Food = "cheese"] -
             -locatedIn-> [Position = [5,8]],
             -exists-> [Quantity = 300 ].
Environment( Food ):: [Food = "fish"] -
             -locatedIn-> [Position = [3,9]],
             -exists-> [Quantity = 300 ].
Environment( Agent ):: [Agent = John] -
             -locatedIn-> [CurrentPosition = [5,5]],
             -typeOf-> [Type = Bear ].
Environment( Agent ):: [Agent = Arthur] -
             -locatedIn-> [CurrentPosition = [16,7]],
             -typeOf-> [Type = Bird ].

Environment( Trajectory ):: [ Trajectory = "Food1"]-
                    -has-> [ List = [ [5,5] , [3,5] , [4,7] , [2,9], [3,11] ] ],
                    -obj-> [ Object =  "honey" ],
                    -typeOf-> [ Type =  Food ].
Environment( Trajectory ):: [ Trajectory = "Food2"]-
                    -has-> [ List = [ [16,7] , [18,5] , [20,6] , [22,6] , [22,7]
, [24,6]]],
                    -obj-> [ Object =  "honey" ],
                    -typeOf-> [ Type =  Food ].
Environment( Trajectory ):: [ Trajectory = "Food3"]-
                    -has-> [ List = [ [16,7] , [15,9] , [17,11], [19,9] , [20,9]
, [20,11]] ],
```

```
                    -obj-> [ Object =  "honey" ],
                    -typeOf-> [ Type =  Food ].
Environment( Trajectory ):: [ Trajectory = "Food4"]-
                    -has-> [ List = [ [16,7] , [15,5] , [13,4] , [14,2], [18,1] ,
[21,3] , [23,3]] ],
                    -obj-> [ Object =  "cheese" ],
                    -typeOf-> [ Type =  Food ].
Environment( Trajectory ):: [ Trajectory = "Agent1"]-
                    -has-> [ List = [ [5,5] , [6,6] , [8,7] , [10,8] , [11,10] ,
[12,8] , [14,7] , [16,7]] ],
                    -obj-> [ Object =  "Agent" ],
                    -typeOf-> [ Type =  Agent ].
```

ONTOLOGY.XML (Please define the Ontology attributes, goals for the Animals and the XML file is created)

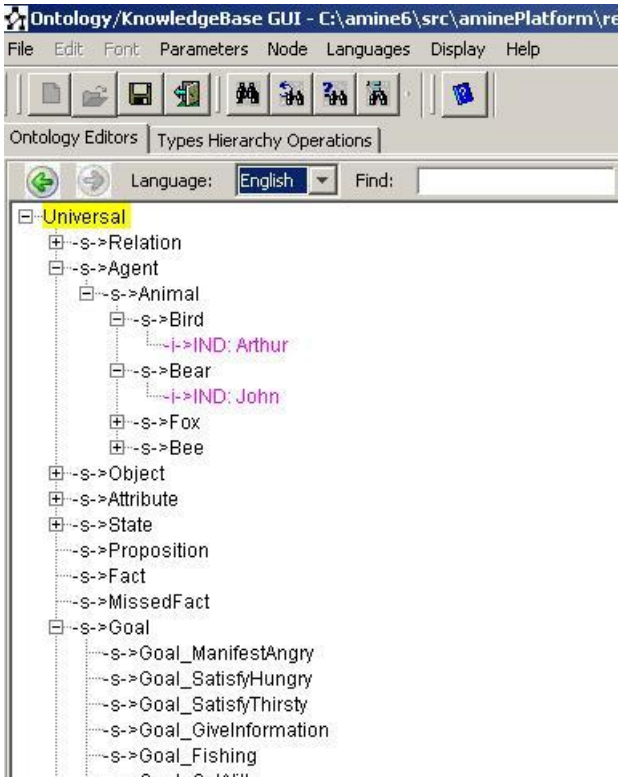


Figure A.1 Ontology definition using AMINE

Appendix III

agentTool Betting Game Code

Bet_PlayerAgent.Java

```
/* Conversation class for Bet_PlayerAgent_R.
* automatically generated by agentTool - agentMom transform ver 0.5
*/

package Poker;

import java.net.*;
import java.io.*;
import afit.mom.*;
public class Bet_PlayerAgent_R extends Conversation
{
	public Component parent; // override parent definition in Agent
	public Object choice;
	public Object bet;
	public Object options;
	public Object amt;
	public Object status;
	public bet_options_Type bet_options;

/*  Constructor for  Bet_PlayerAgent_R.
*  automatically generated by agentTool - agentMom transform ver 0.5
*/
public Bet_PlayerAgent_R(Socket s, ObjectInputStream i, ObjectOutputStream o,
Component c, Message m1){
	super(s, i, o, c, m1);
	parent = c;
}

/* run method - controls state table behavior.
*  automatically generated by agentTool - agentMom transform ver 0.5
*/
public void run()
{	int state = 0;
	boolean notDone = true;

	/* state constant definitions */
	final int StartState = 0;
	final int Get_Choice = 1;
	final int Get_Amount = 2;
	final int Wait_For_Status = 3;
	final int Show_Status = 4;
	final int StartState_out = 5;
	final int Get_Choice_out = 6;
	final int Get_Amount_out = 7;
	final int Wait_For_Status_out = 8;
	final int Show_Status_out = 9;
	//set up
	try {
		while (notDone) {
			switch (state) {
				case StartState :
					state = StartState_out;
					break;
				case StartState_out :
					m = nonblockedReadMessage(input);
					if(m != null){
						// Inform(bet, options)
						if(m.performative.equals("Inform")){
							bet_options =
```

```
(bet_options_Type)m.content;
bet = bet_options.bet;
                                        options = bet_options.options;
                                        state = Get_Choice;
                                }
                        }
                        break;
                case Get_Choice :
                        choice = parent.getChoice(bet, options);
                        state = Get_Choice_out;
                        break;
                case Get_Choice_out :
                        // [choice == raise]
                        if(choice == raise) {
                                state = Get_Amount;
                        }
                        // [choice == call] ^Call()
                        else if(choice == call) {
                                m = new Message();
                                m.performative = "Call";
                                state = Wait_For_Status;
                        }
                        // [choice == see] ^See()
                        else if(choice == see) {
                                m = new Message();
                                m.performative = "See";
                                state = Wait_For_Status;
                        }
                        // [choice == fold] ^Fold()
                        else if(choice == fold) {
                                m = new Message();
                                m.performative = "Fold";
                                notDone = false;
                        }
                        break;
                case Get_Amount :
                        amt = parent.getAmount();
                        state = Get_Amount_out;
                        break;
                case Get_Amount_out :
                        //  ^Raise(amt)
                        m = new Message();
                        m.performative = "Raise";
                        m.content = amt;
                        state = Wait_For_Status;
                        break;
                case Wait_For_Status :
                        state = Wait_For_Status_out;
                        break;
                case Wait_For_Status_out :
                        m = nonblockedReadMessage(input);
                        if(m != null){
                                // Inform(status)
                                if(m.performative.equals("Inform")){
                                        status = m.content;
                                        state = Show_Status;
                                }
                        }
                        break;
                case Show_Status :
                        parent.show(status);
                        state = Show_Status_out;
                        break;
                case Show_Status_out :
                        //
                        notDone = false;
                        break;
        }
    }
```

```
                input.close();
                output.close();
                connection.close();
        } catch (UnknownHostException e) {
                parent.write("** Unknown Host exception in Bet_PlayerAgent_R.");
        } catch (EOFException eof) {
                parent.write("** Server terminated connection in
Bet_PlayerAgent_R.");
        } catch (IOException e) {
                parent.write("** IO Exception in Bet_PlayerAgent_R.");
                e.printStackTrace();
        }
}
} // end class
```

Bet_PokerServer.java

```
/* Conversation class for Bet_PokerServer_I.
* automatically generated by agentTool - agentMom transform ver 0.5
* /

package Poker;

import java.net.*;
import java.io.*;
import afit.mom.*;
public class Bet_PokerServer_I extends Conversation
{
        public Component parent; // override parent definition in Agent
        public Object bet;
        public Object amt;
        public Object status;
        public Object options;
        public bet_options_Type bet_options;

/* Constructor for  Bet_PokerServer_I.
*automatically generated by agentTool - agentMom transform ver 0.5
*/
public Bet_PokerServer_I(Component c, String hostName, int portNum, Object
bet_in, Object options_in)
{
        super(c, hostName, portNum);
        parent = c;

        // save input variables to class attributes
        bet = bet_in ;
        options = options_in ;

// open a new socket connection
        try{
                connection = new Socket(hostName, portNum);
                output = new ObjectOutputStream(connection.getOutputStream());
                output.flush();
                input = new ObjectInputStream(connection.getInputStream());
        }catch(Exception e){System.out.println(e);}
}

/* run method - controls state table behavior.
* automatically generated by agentTool - agentMom transform ver 0.5
*/
public void run()
{
        int state = 0;
        boolean notDone = true;

        /* state constant definitions */
        final int StartState = 0;
```

```
final int Wait_For_Choice = 1;
final int See = 2;
final int Call = 3;
final int Raise = 4;
final int Fold = 5;
final int Show_Status = 6;
final int StartState_out = 7;
final int Wait_For_Choice_out = 8;
final int Show_Status_out = 9;
final int See_out = 10;
final int Fold_out = 11;
final int Call_out = 12;
final int Raise_out = 13;

//set up
try {
       while (notDone) {
              switch (state) {
                     case StartState :
                            state = StartState_out;
                            break;
                     case StartState_out :
                            //  ^Inform(bet, options)
                            m = new Message();
                            m.performative = "Inform";
                            bet_options = new bet_options_Type();
                            bet_options.bet = bet;
                            bet_options.options = options;
                            m.content = bet_options;
                            sendMessage(m, output);
                            state = Wait_For_Choice;
                            break;
                     case Wait_For_Choice :
                            state = Wait_For_Choice_out;
                            break;
                     case Wait_For_Choice_out :
                            m = nonblockedReadMessage(input);
                            if(m != null){
                                   // See()
                                   if(m.performative.equals("See")){
                                          state = See;
                                   }
                                   // Call()
                                   else if(m.performative.equals("Call")){
                                          state = Call;
                                   }
                                   // Raise(amt)
                                   else if(m.performative.equals("Raise")){
                                          amt = m.content;
                                          state = Raise;
                                   }
                                   // Fold()
                                   else if(m.performative.equals("Fold")){
                                          state = Fold;
                                   }
                            }
                            break;
                     case See :
                            parent.update(bet);
                            state = See_out;
                            break;
                     case See_out :
                            //
                            state = Show_Status;
                            break;
                     case Call :
                            parent.update(bet);
                            state = Call_out;
                            break;
```

```
                                case Call_out :
                                        //
                                        state = Show_Status;
                                        break;

                                case Raise :
                                        parent.raiseBet(amt);
                                        parent.update(bet);
                                        state = Raise_out;
                                        break;
                                case Raise_out :
                                        //
                                        state = Show_Status;
                                        break;
                                case Fold :
                                        parent.update(bet);
                                        parent.fold();
                                        state = Fold_out;
                                        break;
                                case Fold_out :
                                        //
                                        state = Show_Status;
                                        break;
                                case Show_Status :
                                        status = parent.getTableStatus();
                                        state = Show_Status_out;
                                        break;
                                case Show_Status_out :
                                        //  ^Inform(status)
                                        m = new Message();
                                        m.performative = "Inform";
                                        m.content = status;
                                        notDone = false;
                                        break;
                        }
                }
                input.close();
                output.close();
                connection.close();
        } catch (UnknownHostException e) {
                parent.write("** Unknown Host exception in Bet_PokerServer_I.");
        } catch (EOFException eof) {
                parent.write("** Server terminated connection in
Bet_PokerServer_I.");
        } catch (IOException e) {
                parent.write("** IO Exception in Bet_PokerServer_I.");
                e.printStackTrace();
        }
}
} // end class
```

Literature Cited / Bibliography

Anonymous. (2001). *AI winter*. Retrieved 2011, from Wikipedia: http://en.wikipedia.org/wiki/AI_winter

Anonymous. (2011). *History of artificial intelligence*. Retrieved 2011, from Wikipedia: http://en.wikipedia.org/wiki/History_of_artificial_intelligence

Bellifemine. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *SOFTWARE—PRACTICE AND EXPERIENCE* , 103 - 128.

Board, J. (2005). *JADE SECURITY GUIDE.* Boston: Free Software Foundation.

Chess, K. a. (2003, Jan). The Vision of Autonomic Computing. *Computer* , pp. 41-50.

dannyweyns. (2009). *Packet world*. Retrieved 2011, from Sourceforge: http://sourceforge.net/projects/packet-world/

Gómez, P. (2005). *INGENIAS Development Kit (IDK) Manual.* Madrid: Universidad Complutense de Madrid.

KABBAJ, P. A. (2009). *http://sourceforge.net/*. Retrieved 2011, from http://amine-platform.sourceforge.net/

Lenica, P. *Toward Agent-based Cooperative Resource Management in a Telecom Operator Grid Platform.* France: France Telecom - R&D Division.

Pervez, M. T. (2010). FMSIND: A Framework of Multi-Agent Systems Interaction during Natural Disaster. *Journal of American Science* , 217-224.

Platon, D. E. (2007). *MODELING EXCEPTION MANAGEMENT IN MULTI-AGENT SYSTEMS.* Japan: The Graduate University for Advanced Studies (SOKENDAI).

René, G. a. (1993). Theories about architecture and performance of MAS. *III European Congress of Psychology* (pp. 1-33). Finland: Groningen University.

Shelley, M. (1818). *The Origin of a Myth*. Retrieved 2011, from Frankenstein films: http://members.aon.at/frankenstein/frankenstein-novel.htm

Torres, V. (2002). *Theoretical Foundations for Agents and Objects in Software Engineering.* Brazil: PUC-Rio, Computer Science Department,.

Wood, M. (2000). *Multiagent Systems Engineering: A Methodology for ANALYSIS AND DESIGN OF MAS.* Ohio: AIR FORCE INSTITUTE OF TECHNOLOGY.

Wooldridge, M. (1997). Agent-Based Software Engineering. IEEE Proceedings on Software Engineering. *IEEE Proceedings*, (pp. 26–37).

Zou, Y. *Using Semantic Web technology in Multi-Agent Systems.* Baltimore: U.Maryland Baltimore County.