



UNIVERSIDAD  
POLITECNICA  
DE VALENCIA

---

# Requirements Modeling for Multi-Agent Systems

---

**María Lorena Rodríguez Viruel**

DIRECTORES

**Dr. Emilio Insfran Pelozo**

**Dr. Luca Cernuzzi**

Master thesis in Software Engineering, Formal  
Methods and Information Systems (ISMFSI)

Departamento de Sistemas Informáticos y  
Computación (DSIC)



Ingeniería del Software  
y Sistemas de Información

February 2011

## Agradecimientos

*A mis directores de tesis Emilio Insfran y Luca Cernuzzi por su esfuerzo, dedicación, confianza y ayuda fundamental para la concreción de este trabajo.*

*A la entidad Itaipu Binacional/Parque Tecnológico Itaipu-Py por la beca otorgada para la realización de los estudios del master.*

*Me gustaría también expresar mi sincero agradecimiento a todo el grupo ISSI, por la ayuda y consejos prestados.*

*A los buenos amigos que forme durante mi estancia en Valencia, por el apoyo incondicional y desinteresado. Por toda la ayuda y cariño que me han dado.*

*Y, por supuesto, el agradecimiento más profundo y sentido va para mi familia, por el ánimo y cariño de siempre. A mis padres, por la formación que me han permitido obtener, por su ejemplo. A Maurizio, por el apoyo, por estar a mi lado para que haya podido realizar y finalizar esta etapa académica.*

---

## Abstract

Different approaches for building modern software systems in complex and open environments have been proposed in the last few years. Some efforts try to take advantage of the agent-oriented paradigm to model/engineer complex information systems in terms of independent agents. These agents may collaborate in a computational organization (Multi-Agent Systems, MAS) by playing some specific roles having to interact with others in order to reach a global or individual goal. In addition, due to the complex nature of this type of systems, dealing with the classical *functional* and *structural* perspectives of software systems are not enough. The *organizational* perspective, that describes the context where these agents need to collaborate, and the *social behavior* perspective, that describes the different “intelligent” manners in which these agents can collaborate, need to be identified and properly specified.

Several methodologies have been proposed to drive the development of MAS (e.g., Ingenias, Gaia, Tropos) although most of them mainly focus on the design and implementation phases and do not provide adequate mechanisms for capturing, defining, and specifying software requirements.

Poor requirements engineering is recognized as the root of most errors in current software development projects, and as a means for improving the quality of current practices in the development of MAS, the *main objective* of this work is to propose a requirements modeling process to deal with software requirements covering the functional, structural, organizational, and social behavior perspectives of MAS.

The requirements modeling proposed is developed within the model-driven engineering context defining the corresponding metamodel and its graphical syntax. In addition, a MAS requirements modeling process is specified using the Object Management Group’s (OMG) Software Process Engineering Metamodel (SPEM). Finally, in order to illustrate the feasibility of our approach, we specified the software requirements of a strategic board game (the Diplomacy game).

## Resumen

Diferentes enfoques se han propuesto en los últimos años para la construcción de sistemas de software modernos en entornos complejos y abiertos. Algunos esfuerzos intentan aprovechar el paradigma orientado a agente para modelar/desarrollar sistemas de información complejos en términos de agentes independientes. Estos agentes pueden colaborar en una organización computacional (Sistemas Multi-Agente, SMA) llevando a cabo roles específicos e interactuando con otros para alcanzar un objetivo global o individual. Además, debido a la naturaleza compleja de este tipo de sistemas, tratar con las clásicas perspectivas *funcionales* y *estructurales* del sistema de software no es suficiente. También es necesario identificar y especificar correctamente la perspectiva *organizacional*, que describe el contexto donde estos agentes necesitan colaborar, y la perspectiva de *comportamiento social*, que describe los diferentes modos “inteligentes” en que estos agentes pueden colaborar.

Varias metodologías se han propuesto para guiar el proceso de desarrollo de un SMA (ej., Ingenias, Gaia, Tropos) sin embargo muchas de estas propuestas se centran principalmente en las fases de diseño e implementación y no proveen mecanismos apropiados para capturar, definir, y especificar los requisitos del software.

Una aplicación pobre de la ingeniería de requisitos es reconocida como la raíz de la mayoría de los errores en los desarrollos de software actuales, por tanto para mejorar la calidad de las prácticas actuales en el desarrollo de SMA, el *objetivo principal* de este trabajo es proponer un proceso de modelado de requisitos para tratar los requisitos de software cubriendo las perspectivas *funcional*, *estructural*, *organizacional* y de *comportamiento social* de un SMA.

La propuesta de modelado de requisitos es desarrollada en el contexto del desarrollo dirigido por modelos definiendo los metamodelos correspondientes y su sintaxis gráfica. Además, se especifica el proceso de modelado de requisitos de SMA utilizando el *Software Process Engineering Metamodel* (SPEM) propuesto por el *Object Management Group* (OMG). Por último, con el fin de ilustrar la factibilidad de nuestro enfoque, se especifica los requisitos de software de un juego estratégico de tablero (Diplomacia).

---

## Resum

Diferents aproximacions han sigut proposades en els últims anys per a la construcció de sistemes de programari moderns en entorns complexos i oberts. Alguns d'aquests esforços tracten d'aprofitar el paradigma orientat a agents per tal de modelar/desenvolupar sistemes d'informació complexos en termes d'agents independents. Aquests agents poden col·laborar en una organització computacional (Sistemes Multi-Agent, SMA) adoptant rols específics e interactuant amb altres per tal d'aconseguir un objectiu global o individual. A més a més, donada la complexitat d'aquest tipus de sistema, no és suficient amb el tractament de la perspectiva funcional i estructural del sistema de programari. També cal identificar i especificar correctament la perspectiva organitzacional, la qual descriu el context on aquests agents necessiten col·laborar, i la perspectiva de comportament social, que descriu els diferents modes "intelligents" de col·laboració.

Diverses metodologies han sigut proposades per guiar el procés de desenvolupament d'un SMA (ex. Ingenias, Gaia, Tropos). No obstant això, moltes d'aquestes es centren principalment en les fases de disseny i implementació, i no proveeixen de mecanismes adequats per a capturar, i especificar els requisits del programari.

Donat que la poca aplicació de l'enginyeria de requeriments ha sigut principalment identificada com l'arrel de la majoria dels errors en el desenvolupament de programari, i amb la mirada posada en la millora de la qualitat de les pràctiques que s'apliquen actualment en el desenvolupament de sistemes multi-agent, l'objectiu principal d'aquest treball és proposar un procés de modelatge de requisits per a suportar la gestió i tractament de requisits de programari, cobrint les perspectives funcional, estructural, organitzacional i de comportament social d'un SMA .

El model de requisits proposat es desenvolupa en el context de l'enginyeria dirigida per models, definint metamodels corresponents i la sintaxi gràfica corresponent. A més, s'especifica el procés de modelatge de requisits per a sistemes multi-agent emprant el Software Process Engineering Metamodel (SPEM), proposat pel Object Management Group (OMG). Per últim, amb l'objectiu d'illustrar la viabilitat de la nostra aproximació, s'especifiquen els requisits de programari d'un joc de taula (Diplomàcia).

# Contents

Chapter 1 Introduction .....	1
1.1. Motivations .....	1
1.2. Objectives.....	4
1.3. Research context.....	4
1.4. Structure of the thesis.....	5
Chapter 2 Fundamentals and Technological Areas.....	7
2.1. Requirements Engineering .....	7
2.1.1 Definitions .....	7
2.1.2 RE Activities.....	10
2.2. Multi-Agent Systems.....	12
2.2.1 Definitions .....	13
2.2.2 Common abstractions .....	15
2.3. Model Driven Software Development.....	16
2.3.1 Models – The foundation of MDSD .....	17
2.3.2 Domain Specific Modeling Language (DSML) .....	18
2.3.3 Meta-Object Facility (MOF) .....	21
2.3.4 Model Driven Architecture (MDA).....	22
2.4. Technological Areas .....	23
2.4.1 Eclipse .....	23
2.4.2 EMF .....	23
Chapter 3 Perspectives for Multi-agent Systems.....	25
3.1. Functional Perspective .....	25
3.2. Structural Perspective.....	26
3.3. Organizational Perspective .....	26
3.4. Social Behavior Perspective .....	28
Chapter 4 Related Work .....	31
4.1. Agent-Oriented Methodologies .....	31

---

4.2. Agent-oriented Methodologies in the Context of Model Driven Development .....	35
4.3. Modeling requirements for multi-agent systems .....	37
4.4. A Systematic Review of Requirements Engineering in the Multi-Agent Systems .....	39
4.5. Discussion .....	40
Chapter 5 Modeling Requirements for Multi-Agent Systems .....	41
5.1. Proposal .....	41
5.1.1 Requirements Definition .....	42
5.1.2 Requirements Specification .....	44
Chapter 6 Requirements Metamodel for Multi-Agent Systems .....	47
6.1. Organizational Model .....	49
6.2. Role Model .....	50
6.3. Goal Model .....	52
6.4. Domain Model .....	54
6.5. Social Behavior Model .....	55
6.6. Environment Model .....	58
6.7. Organizational Rules Model .....	59
Chapter 7 Requirements for MAS Modeling Process .....	61
7.1. Introduction to SPEM 2.0 .....	61
7.2. Disciplines .....	65
7.2.1 Requirements Definition Discipline .....	65
7.2.2 Requirements Specification Discipline .....	66
7.3. Process .....	67
7.3.3 Requirements Definition .....	68
7.3.4 Requirements Specification .....	69
Chapter 8 Case Study .....	71
8.1. Diplomacy Game .....	71
8.2. Diplomacy Game with Agents .....	73
8.2.1 Requirements Definition .....	73
8.2.2 Requirements Specification .....	76

8.3. Discussion .....	95
Chapter 9 Conclusions.....	97
9.1. Contributions .....	97
9.2. Future Work.....	101
9.3. Related Publications .....	102
Bibliography .....	106



---

## List of figures

Figure 2-1 Canonical View of an Agent-based System [39].....	15
Figure 2-2 Cost prediction for DSL-based methodologies [21] .....	19
Figure 2-3 Example of MOF layers .....	21
Figure 5-1 Models of the proposal and its relationships .....	42
Figure 6-1 Requirements metamodel for MAS .....	48
Figure 6-2 Organizational Model.....	49
Figure 6-3 Organizational Model Graphical Syntax .....	50
Figure 6-4 Role Model .....	51
Figure 6-5 Role Model Graphical Syntax: Refinement Tree .....	52
Figure 6-6 Role Model Graphical Syntax: Inheritance relation diagram.....	52
Figure 6-7 Goal Model .....	53
Figure 6-8 Goal Model Graphical Syntax .....	53
Figure 6-9 Domain Model .....	54
Figure 6-10 Domain Model Graphical Syntax.....	55
Figure 6-11 Social Behavior Model .....	56
Figure 6-12 Social Behavior Model Graphical Syntax: Social Behavior Diagram.....	57
Figure 6-13 Social Behavior Model Graphical Syntax: Activity Diagram.....	57
Figure 6-14 Environment Model.....	58
Figure 6-15 Environment Model Graphical Syntax .....	59
Figure 6-16 Organizational Rules Model.....	60
Figure 7-1 SPEM modeling primitives .....	62
Figure 7-2 Disciplines of the Requirements Modeling Process for MAS.....	65
Figure 7-3 Requirement Definition Discipline .....	66
Figure 7-4 Requirement Specification Discipline .....	67
Figure 7-5 Requirements modeling process overview.....	68

Figure 7-6 Requirements Definition activity decomposed into tasks and artifacts .....	69
Figure 7-7 Requirements Specification activity decomposed into tasks and artifacts .....	70
Figure 8-1 Diplomacy Game Refinement Tree .....	74
Figure 8-2 Diplomacy Game Inheritance Diagram .....	75
Figure 8-3 Diplomacy game Domain Model .....	75
Figure 8-4 Social behavior diagram of the Initial Phase sub-organization .....	76
Figure 8-5 Social behavior diagram of the Diplomatic Phase sub-organization .....	77
Figure 8-6 Social behavior diagram of the Writing Order Phase sub-organization .....	78
Figure 8-7 Social behavior diagram of the Order Resolution Phase sub-organization .....	79
Figure 8-8 Social behavior diagram of the Retreat and Disband Phase and Gaining and Loosing Units Phase sub-organizations .....	81
Figure 8-9 Activity Diagram of the Start the Game goal .....	83
Figure 8-10 Activity Diagram for the goals Make Alliance and Control Negotiation Session .....	84
Figure 8-11 Activity Diagram for the goals Set Strategy and Control Negotiation Session .....	85
Figure 8-12 Activity Diagram for the goals Make Order and Manage Order .....	86
Figure 8-13 Activity Diagram for the goals Resolve Order Conflicts and Follow Order .....	87
Figure 8-14 Activity Diagram for the goals Make Retreats, Manage Retreats and Follow Retreats.....	88
Figure 8-15 Activity Diagram for the goals Make Adjustments, Manage Adjustments and Follow Adjustments.....	89
Figure 8-16 Activity Diagram for the goals Determine Winner .....	90
Figure 8-17 Diplomacy game Environment Model.....	91

---

## List of tables

Table 6-1 Organizational Rules Model Graphical Syntax .....	60
Table 7-1 Subset of elements to model processes in SPEM 2.0 .....	63
Table 8-1 Organizational Rules Model .....	92

# Chapter 1

## Introduction

Different approaches for building modern software systems in complex and open environments have been proposed in the last few years. Some efforts try to take advantage of the agent-oriented paradigm to model/engineer complex information systems in terms of independent agents. However, most of these methodologies focus on the design and implementation phases within the software life cycle, giving less support to the requirements activity, although poor requirements activity is recognized as the root of most errors in current software systems and has a direct impact on the quality and cost of final product.

In addition, due to the complex nature of this type of systems, dealing with the classical functional and structural aspects of software systems are not enough. The organizational aspect, where these agents need to collaborate, and the social behavior aspect, that these agents need to incorporate, must also be properly identified and specified.

As a means for improving the quality of current practices in the development of multi-agent system, the main goal of the master thesis is to define a requirements modeling process to deal with user and software requirements, emphasizing both the social behavior and the organizational aspects as key aspects for the development of multi-agent systems.

In addition, model-driven software development techniques are adopted with the aim of facilitating the integration of the requirements models generated in our proposal with analysis and design models, and thus facilitate maintainability, reusability, interoperability and improved adaptation of technological change.

### 1.1. Motivations

Requirements Engineering (RE) is responsible for bridging the gap between the informal world of the client and the formal world of software engineering. RE plays a key role in software development.

In fact, the inadequate management of system requirements is a major cause of problems in software development [1]. RE is a branch of software engineering and includes a set of activities concerning the elicitation, specification, analysis, validation and verification, and management of system requirements. In particular, the purpose of requirements identification and specification is to capture the main features of the software system-to-be in a precise and accurate manner. The requirements specification should permit the representation of system requirements so that any potential users can review and understand them. However, the notation used in such a representation is expected to be sufficiently accurate to serve as a basis for subsequent phases [40].

New challenges have appeared in the field of software engineering with the demand for systems in which complexity is observed not only in magnitude but also in the dynamic characteristics of the systems and changeable environment in which they operate. Such open and dynamic systems stress the need for mechanisms with which to represent interaction, pro-activeness, negotiation, etc. among software components.

In response to these new challenges, Agent Oriented Software Engineering (AOSE) has arisen as a new discipline for software development. An agent is characterized as having its own flow control which aims to achieve goals within an environment that may be unpredictable, and in which it should ensure the achievement of its goals by adapting when necessary. These agents often operate in environments in which there are also other agents. A Multi-Agent System (MAS) is a specific type of system that is composed of multiple intelligent agents that interact with each other to achieve certain objectives. These systems can be used to solve problems that it is difficult or impossible for a monolithic or a single agent system to resolve.

In recent years, various methodologies have been proposed to guide the development of MAS, such as Gaia [73], Ingenias [31], Tropos [29], etc. However, despite the importance of the requirements phase in the development of software systems, many of the proposed methodologies for the development of MAS do not adequately cover the RE phase [13], focusing mainly on the design and implementation phases. Moreover, a recent study on the

application of RE techniques in the development of a MAS [7] found that 79% of the current methodologies for MAS development use RE techniques which have been adapted from other paradigms (object orientation, knowledge engineering, etc.) [7]. However, these techniques and notations may not be sufficient to cover the nature of MAS, since these systems need, along with their functional, structural, or organizational properties, characteristics that are not normally necessary in conventional software systems such as proactivity, adaptability, collaboration, truth, or disposition. Therefore, the need for new methods and techniques that enable the appropriate acquisition and treatment of MAS requirements thus arises.

Model-driven software development (MDSD) is a proposal to maximize productivity, enhancing aspects such as software reusability, interoperability and improved adaptation of technological change. One of the most popular approaches is that of Model Driven Architecture (MDA) [47] software development, which is based on the set of standards proposed by the OMG [54]. MDA advocates taking models as the main artifacts of software development as a series of model transformations. It is worth noting that this technology has recently been introduced into AOSE literature and merged with software agent technology to support the development of MAS [57]. In this scenario, the use of a MDSD approach to model MAS requirements will benefit our work from the advantages of the application of those techniques mentioned above, along with bridging the gap between MAS requirements, captured as requirements models, with analysis and design models. Also, is useful to enhance the potential, improve the quality and efficiency of our work, and consequently allow the proposal to be widely adopted by researchers and practitioners in the software engineering community.

Finally, agent technology is useful in many complex domains: e-commerce, health, stock market, manufacturing, games, etc. In particular, we are interested in the game development domain since it comprises a set of characteristics such as collaboration, negotiation, trust, reputation, etc., which specially can be dealt with a MAS. According to Google Trends and the ESA annual report [22], games development is one of the business markets that has undergone most growth in the last few years. In addition, the agent-oriented paradigm

is one of the most promising for modeling such business market due to the social behavior characteristics (negotiation, cooperation, etc.) of the agents and the complexity that MASs can support.

## 1.2. Objectives

The main objective of this master thesis is to propose a requirements modeling process to deal with user and software requirements emphasizing both the social behavior and the organizational aspects as key aspects for the development of MAS. In order to obtain the main objective, this thesis is broken down into the following research tasks:

- i) perform a literature review with which to investigate current techniques, methods, and methodologies to develop MAS;
- ii) perform a literature review with which to investigate current RE techniques, methods, and methodologies with particular emphasis on those techniques more appropriate to identify and specify requirements for complex and dynamic systems;
- iii) study and define the perspectives needed to adequately represent a MAS;
- iv) define the proposal for requirements modeling of MAS based on previous studies in the related topics, and covering the perspectives defined;
- v) extend the proposal towards a MDSO approach, enabling usability evaluations on model integration with other MAS methodologies at different stages of development;
- vi) specify the development process through a process modeling language, to precisely specify how to use the proposed method and artifacts to be generated;
- vii) illustrate the feasibility of the approach by applying the requirements modeling process to the development of the strategic board Diplomacy Game [23];

## 1.3. Research context

This investigation work started in 2007 as part of the final thesis for engineering degree at the "Nuestra Señora de la Asunción" Catholic University in Asuncion, Paraguay. The early work was included in the following projects:

- “Diseño y Modelado de Sistemas Agentes y Multi-Agentes: Propuestas de Mejoras”, funded for the “Nuestra Señora de la Asunción” Catholic University;
- MENSA (“Methodologies for the Engineering of complex Software systems: Agent-based approach”), cooperation project with three Italian universities (Alma Mater Studiorum - Università di Bologna (Cesena branch), Università degli Studi di Modena e Reggio Emilia, and Università degli Studi di Trento).

Currently we have continued research work in the ISSI group (Ingeniería del Software y Sistemas de Información), and under a scholarship in the “College Scholarship Program and Support of the Scientific and Technological Production”, in the context of the Social Responsibility Program of the Itaipu Binacional/Parque Tecnológico Itaipu-Paraguay.

Resulting in this master thesis and as a contribution as invited researcher in the Multi-modeling Approach for Quality-Aware Software Product Lines (MULTIPLE) Project with ref. TIN2009-13838 (October 2009 - September 2013) funded by the Ministry of Science and Innovation (Spain). Specifically, working in the domain of games, using MAS.

## 1.4. Structure of the thesis

The structure of this master thesis is as follows:

Chapter 2 gives an overview of the background concepts of this work. First, it defines the concepts necessary to understand the context of AOSE. In addition, it includes the most important definitions in the area of RE necessary for the understanding of this proposal. Also, it includes a description of the MDSD context, focusing on MDA, the OMG proposed architecture, and standards. Finally, it introduces the technology space related to this thesis: the Eclipse environment, and Eclipse Modeling Framework.

Chapter 3 describes the different perspectives proposed to be considered when modeling MAS. The main purpose of the definition of perspectives for MAS is to have a clear idea of which common and special characteristics to capture in this kind of systems.

Chapter 4 presents the result of a literature review with which to study the existing works in the interrelated field of RE, Agent-



Oriented approaches, and MDSO. In particular, presents the most relevant agent-oriented methodologies found in the literature, approaches of agent-oriented methodologies in the MDSO context, proposals of requirements modeling for MAS, and finally, the results of a systematic review of the use of RE in the development of MAS.

Chapter 5 presents the proposal of a requirements modeling for MAS. In particular, it describes the proposal phases, and outlines the models needed to describe the problem in more specific aspects to form the different perspectives of the MAS.

Chapter 6 continues with the proposal presenting the metamodel defined for the proposed requirements modeling for MAS. In particular we comment the metamodels developed for the proposed requirements for MAS modeling and the corresponding graphical syntax.

Chapter 7 completes the proposal describing the methodological guide: a description of the successive steps, activities, and guidelines for identify and specify the requirements of a MAS by using the proposal. The OMG standard SPEM 2.0 is used for this description.

To validate de feasibility of the proposal in Chapter 8 a case study in the game domain is presented, the Diplomacy Game, following the process presented in the previous chapter.

Finally, Chapter 9 concludes with the main contributions of the thesis, introduces future research work, and presents the related publications.

# Chapter 2

## Fundamentals and Technological Areas

Great efforts in AOSE focus on the definition of methodologies for the analysis, design and development of MAS. However, not all AOSE methodologies proposed so far considered the requirements elicitation, requirements are often neglected or treated only superficially. However, the value of good RE, and the importance of doing well, has dramatically increased with the size and complexity of software systems. This is because a critical factor for the successful development of such systems is to understand what customers need and want the system to do, and RE is the area that focuses on the identification and characterization of exactly that. Also, MDSD is a proposal to maximize productivity, enhancing aspects such as software reusability, interoperability and improved adaptation of technological change. For these reasons, this work focuses on RE for agent-oriented paradigm in the context of MDSD.

This chapter explains all the concepts, standards and tools that are directly related to the work proposed along the thesis. Specifically, we present the paradigm of RE, the paradigm of MAS, MDSM and the technology related to the work.

### 2.1. Requirements Engineering

To address the problem of lack of coverage of a requirements phase by current agent-oriented methodologies, first an analysis of what RE currently offers is presented. It is presented below some definitions found in the literature and a classification of the main activities involved in the process of RE.

#### 2.1.1 Definitions

Before going into detail about what is RE, it is important to give a definition of the meaning of requirements, to be clear about what you have to identify, model and analyze. There are many definitions of

requirements, the software engineering glossary of the IEEE [34] defines requirement as:

*“(1) A condition or capability needed by a user to solve a problem or achieve an objective; (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; (3) A documented representation of a condition or capability as in (1) or (2).”*

*“The set of all requirements forms the basis for further development of the system or system component.”*

The requirements cover not only the functionality of a system, but also address non-functional issues (e.g., performance, interface, reliability requirements, etc.), design constraints (e.g., operating in conjunction with existing hardware or software), and implementation constraints (e.g., cost, need to program in Java, etc.). These requirements and restrictions must be represented in a manner that meets the needs of different users. Moreover, some of the main purposes of the RE activities are: reach agreement on the requirements, provide a basis for software design, and provide a benchmark for software validation [10].

RE discipline plays a major role in the software production process, since it addresses a key problem: the definition of what is desired to produce. Its main task is to generate correct specifications that clearly, unambiguous, consistent and compact, describe the behavior of the system thus seeks to minimize problems related to systems development.

The need for proper implementation of RE and the consequences of not doing it correctly, are extremely important. In many cases the specification of requirements seems trivial, but it is probably the stage of development, if its bad design, leads to failure than any other stage. Moreover, the benefits of good RE include: the agreement between developers, customers and users about the work to be done and the system acceptance criteria, a sound basis for resource estimation, improving the usability of the system, maintenance, etc.

In the literature we find different definitions that provide interesting concepts and new perspectives or serve to reaffirm the existing ones.

*“Software requirements engineering is the discipline for developing a complete, consistent unambiguous specification – which can serve as a basis for common agreement among all parties concerned – describing what the software product will do (but not how it will do it, this is to be done in the design specification).” [9]*

*“Requirements engineering is the process by which the requirements stated by customers, whether spoken or written, are transformed to precise, unambiguous, consistent and complete specifications of the system behavior, including functions, interfaces, performance and limitations”. [64]*

In order to highlight its complexity, [51] discuss one of the clearest definitions of requirements engineering applied to software:

*“Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.” [74]*

It highlights the importance of “real-world goals” that motivate the development of a software system. The reference to “precise specifications” provide the basis for analyzing requirements, validating that they are indeed what stakeholders want, defining what designers have to build, and verifying that they have done so correctly upon delivery. Also the definition refers to specifications’ “evolution over time and across software families”, emphasizing the reality of a changing world and the need to reuse partial specifications, as engineers often do in other branches of engineering.

RE is a complex, human-centered process, which comprises a series of questions such as, the approach used, the number of steps needed to complete the process, the interactions necessary to establish sufficient requirements, the definition of how much is enough, the process ‘participants, the role of each participant, the form of communication between them, the representation to be used for documentation, validation of knowledge gained, among others. This is why the tools and techniques used in RE draw upon a variety

of disciplines, and the requirements engineer may be expected to master skills from a number of different disciplines.

This phase of the software development process encompasses a set of activities that usually take place within a social context, therefore it is important to recognize that the system's success depends upon the understanding of, what customers need, how technology can alter their relations, facilitate negotiations and improve the communication of their needs to developers. Also, as already mentioned the requirements change during the development process and evolve once the system is already running. Because of this is that several approaches including models, languages, methodologies and tools have been developed to support the activities of requirements [51].

### **2.1.2 RE Activities**

The activities of RE have been classified by different authors in different ways [7], [16], [20]. Not having a clear agreement, a classification that we consider simple and yet interesting is presented in [14], and covers three main aspects: elicitation, modeling, analysis, validation and verification, and requirements managements. This decomposition must be considered as a description of the tasks that must be carried out, but they do not necessarily have to be performed in an order or at different periods of time. Different processes, which are abstract descriptions of how to run this set of activities describing the engineers' behavior, define different flow of activities. Generally, these activities are more likely to occur simultaneously throughout the RE process.

We discuss the most important activities in relation to this work, including the main notations and techniques related to each of them.

#### **2.1.2.1 Elicitation**

Requirements elicitation is usually the first stage of the RE process. It covers the activities to understand the goals, objectives and motivations for building the proposed software system. This includes establishing the limits of the system, defined at a high level. This limits are of the operational environment of the system. There are many techniques and notations proposed to obtain this information:

- Stakeholders identification. The activity of acquiring the system requirements may include the identification of stakeholders. The term stakeholder refers to those who can affect or be affected by the activities of a company. A stakeholder can be a customer, developer or user.
- Goals. Denote the objectives a system must achieve. Identify high-level goals early in the development of the system is a crucial task in software development. The goal-oriented elicitation is an activity where high-level requirements, as business goals, are refined into lower-level goals, as technical goals, that eventually will be operationalized in a system.
- Scenarios. Is another type of notation, consisting of partial and specific descriptions of system behavior in a given situation.
- Non-functional requirements. Is another type of notation, also known as quality requirements, since are a set of quality characteristics that must be taken into account during the development of a software system.
- Metaphors. Allow us to structure concepts from others concepts.

### **2.1.2.2 Modeling**

The modeling allows expressing the requirement specification in terms of one or more models. These models tend to be more accurate, complete and clear than the models used for the requirements elicitation. The process of creating accurate models helps to identify details that were not identified in the requirements elicitation activity. Modeling notations help to establish limits on levels of abstraction in the descriptions of requirements, providing a vocabulary and structural rules that match the entities, relationships and constraints of the problem behavior being modeled. Among the modeling notations are:

- Object modeling. Propose a description of a set of states, usually finite, where each state represents a configuration of objects.
- Behavioral modeling. Provide abstract descriptions of expected behavior of the system.

### **2.1.2.3 Analysis**

These techniques include efforts to assess the quality of the requirements specification. Some analyze malformation errors of the specification, which is understood as ambiguity, inconsistency or

completeness. Other techniques discussed anomalies such as unknown interactions between requirements, potential obstacles to meeting the requirements and reasoning lost. This activity will include techniques such as: checklists, consistency analysis, conflict analysis, risk management and analysis of variability.

#### **2.1.2.4 Validation and Verification**

The requirements validation ensures that the models and documentation correctly express the needs of stakeholders. The validation is usually done subjectively typically, due to informal requirements documentation. This type of validations requires the user to actively participate in the validation process, reviewing the artifacts directly.

In cases in which the requirements specification was made formally, verification techniques can be applied to verify that it meets the needs of stakeholders. Some verification techniques are:

- Model checking. Check the behavior of the model against a temporal logic property in execution traces.
- Model satisfiability. It is verified that there are valid instances of restricted object models, and that the object models operations preserve the invariant.

#### **2.1.2.5 Management**

Requirements management is an activity that includes several techniques to manage requirements, including the evolution of requirements over time. An interesting issue is the tools and techniques that give partial support to the task of identifying and documenting the traceability relationships between requirements artifacts and artifacts of design. The requirements management activities also include techniques for determining the maturity and stability of acquired requirements, so that the requirements that may change can be isolated.

## **2.2. Multi-Agent Systems**

Continuing with the background study of the proposal of this thesis, we present the basic concepts and definitions needed to know about the area of AOSE. It is presented below some definitions found in the literature and the common abstractions used to model a MAS.

### 2.2.1 Definitions

AOSE was born as a new paradigm of software engineering with which to design and develop complex software system. Different approaches are based on diverse definitions of an agent or MAS. We next examine some of the most relevant agent definitions in the literature.

One key characteristic that differentiates agents from an ordinary program is that the agent must be autonomous. Several definitions of agents include this characteristic, among which we mention:

*“Most often, when people use the term ‘agent’ they refer to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist.” [62]*

*“An agent is an entity that senses its environment and acts upon it.” [61]*

*“An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, in pursuit of its own agenda and so as to effect what it senses in the future.” [26]*

Although not stated explicitly, Russell’s definition implies the notion of autonomy as the agent will act in response to perceiving changes in the environment. The other definitions explicitly state autonomy. But all definitions add some other characteristics, among which interaction with the environment is mentioned by most.

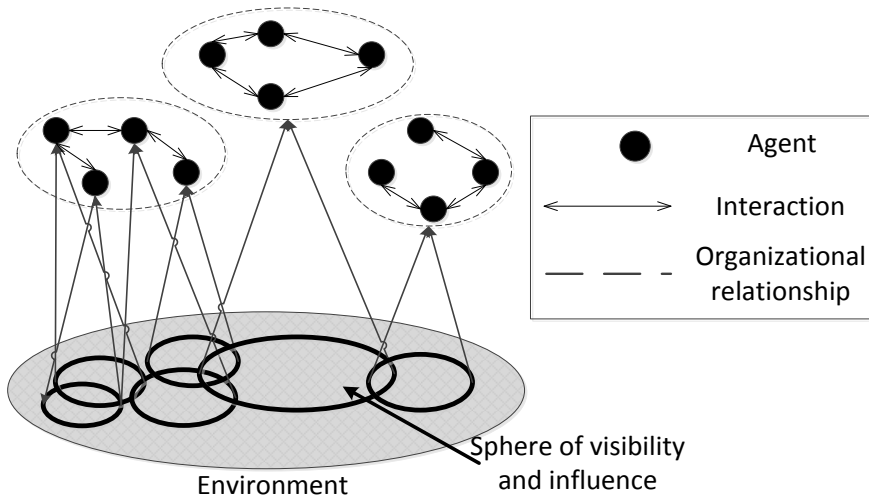
One of the most comprehensive definitions of agents is the one given by Wooldridge and Jennings in [70]. A MAS is a system composed of multiple autonomous software entities (agents) that interact to achieve their goals. They propose the following features for MAS:

- *autonomy*: agents operate without human intervention, and have some control over their actions and internal state;
- *social ability*: agents interact with other agents and possibly humans via some kind of communication language;
- *reactivity*: agents perceive their environment and respond to changes;
- *pro-activity*: agents do not simply act in response to changes in their environment, and are also able to act on their own initiative to achieve their goals.



A MAS by nature represents a decentralized distributed application environment where each agent maintains some level of control or influence in the environment [71], [53]. In a MAS, each agent is aware that it does not possess a global view of the problem and that it cannot solve the problem by itself, thus relying on interaction and coordination with others. It is however still programmed to operate autonomously to compete for satisfaction of its own self-interests, which it believes are benevolent to the goals of the overall group. Fundamental to a MAS environment is this ability for agents to demonstrate social interaction with other agents. As in human social contexts, how agents go about interaction depends on their role and the relationship they have with the target agent. This relationship is described by [71]. Such awareness of the society within which the MAS is operating requires knowledge of the organizational context and structure in order for agents to form productive interactions at runtime with other agents within its environment. This human organization metaphor is proposed in [73] in with:

- A software system is conceived as the computational instantiation of a group of interacting and autonomous individuals (agents);
- Each agent can be seen as playing one or more specific roles: it has a well-defined set of responsibilities or goals in the context of the overall system and is responsible for pursuing these autonomously;
- Interactions are no longer merely an expression of interdependencies, and are rather seen as a means for an agent to accomplish its role in the system. Therefore, interactions are clearly identified and localized in the definition of the role itself, and they help characterize the overall structure of the organization and the position of the agent in it;
- The evolution of the activities in the organization, deriving from the autonomous execution of agents and from their interactions, determines the achievement of the application goal, whether an a priori identified global goal, or a goal related to the satisfaction of individual goals, or both.



**Figure 2-1** Canonical View of an Agent-based System [39]

In summary, the essential concepts of agent-based computing are agents, high-level interactions, social relationships, and organizational relationships (see Figure 2-1). It can be seen that there can be numerous agents wherein specific agents can interact amongst themselves and/or have the same sphere of visibility and influence. There can also be agents who act independently without any interaction of other agents and unique sphere of influence.

### 2.2.2 Common abstractions

Within the AOSE area multiple methodologies have been proposed to guide the MAS development process, such as: Gaia [73], Tropos [29], INGENIAS [31], MaSE [20], MASSIVE [45], etc. Taking into account several of this methodological proposals we can see that different abstractions or terms have been proposed for the characterization of the MAS, the abstractions most commonly identified are:

- *Agents*: autonomous and proactive software entities, which achieve their objectives by interacting with each other and are in a particular environment;
- *Actors*: It is an abstraction of autonomous behavior, internal or external to the system, which has some interest on it and helps define the roles;

- *Roles*: Define the behavior of the agent, and have associated goals and specific tasks to be carried out within the context of the organization;
- *Goals*: Define the objectives of both the general system of each actor. Each goal may relate to functional aspects (associated with the services) or functional (associated with quality of service);
- *Tasks*: A structured set of activities needed to achieve a goal;
- *Restrictions*: The restrictions are that allow us to define the desired behavior for both the organization and for each agent;
- *Interaction between agents*: Typically, the agents operate within a context in which they need to cooperate, compete or communicate solely with them to achieve their own goals;
- *Interactions with the environment*: Agents are typically in an environment with which they may have to interact (detect and affect) depending on their roles, and their current status;
- *Resources*: These are specific components of interaction with the environment;

Abstractions that characterize the MAS should be identified based on clients requirements. That is why it is necessary the requirements phase to carry out an analysis of client needs, aiming to capture key elements that allow us to identify the abstractions that define the system.

### **2.3. Model Driven Software Development**

Model Driven Software Development (MDSD) is a software development approach based on modeling. A model can define the functionality, structure and / or behavior of a system. The models allow working at a level of abstraction closer to the domain concepts, instead of focusing on platform-oriented concepts as with traditional software development. The objective of this proposal is to maximize productivity, increasing the interoperability between systems, facilitating the reuse and adaptation of technological change.

Moreover any software artifact is considered a model or model element. While previous approaches used models for documentation or communication of ideas, they are first class entities throughout the whole model-driven engineering life cycle [36]. MDSD is an open and integrative approach not tied to a special standard, therefore different implementations exist.

MDSD tries to capture what is often expressed in an informal way as formal model based specifications. Key concepts to mitigate current software engineering problem are models, metamodels, technology spaces, domain specific modeling languages and different kinds of model transformation like model to model or model-to-text.

### 2.3.1 Models – The foundation of MDSD

Models are the key artifact of MDSD. For that reason the term model is defined first. A definition from the computer scientists Gerbé and Bézivin [37] is used:

*“A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system.”*

While this serves as a starting point, Kleppe et al. [42] gives a definition even more directed to MDSD:

*“A model is a description of a (part of) systems written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.”*

The word “meta” is Greek and means “above”, therefore the term metamodel can be interpreted as a model describing another model. To understand the term metamodel a simple analogy to languages is drawn. A language consists of words whose combination is constraint by a grammar. If a sentence in a language is seen as one possible model, the definition of its structure, the grammar, can be seen as its metamodel. Earlier it was said that in MDSD a metamodel defines how a model can lookalike, this can be more precisely formulated as: a metamodel defines the constructs and rules usable to create a class of models. This is consistent with the following definition:

*“A metamodel is a model of a set of models.” [38]*

*“A meta-model is a model that defines the language for expressing a model.”[54]*

Notice that a metamodel is itself a model. If this is true the metamodel has also to be defined in a “language” (needs a metamodel). For that reason the metametamodel is introduced,

allowing specifying metamodels. The question which can be raised is how the metamodel is defined. To avoid an infinite stacking of meta levels, metamodels are often specified self-reflexive and therefore the metamodel of the metamodel is the metamodel itself.

Most approaches implementing MDSM define a three level meta stack - model, metamodel, metamodel. A metamodel can be used to clearly define a class of models and the metamodel should allow the specification of all possible metamodels including itself. Therefore one metamodel should be enough.

The one metamodel needs still to be defined. Currently different models compete (e.g. [54], [41], [43], [49]) and different approaches have already been made to bridge metamodels to instance (e.g. [27], [47]). Being the most important the MetaObjectFacility (MOF) [54] and Model Driven Architecture (MDA) [47], respectively.

### **2.3.2 Domain Specific Modeling Language (DSML)**

The ideas behind Domain Specific Modeling Languages (DSML) and Domain Specific Modeling are similar to the Domain Specific Languages (DSL) concepts presented later but applied to the world of models. Whereas in DSLs the language, its textual representation, is domain specific, in DSM the models are domain specific. In comparison with DSLs the representation is almost in every case graphical. The gained benefits are similar, too.

UML according to its name a unified modeling languages tries to offer a set of different model types able to describe every imaginable domain. The recent version of UML allows the modification by profiles, enabling specialization. Of greater importance is the fact that UML diagram types are now based on a single metamodel (MOF).

#### **2.3.2.1 Domain Specific Language (DSL)**

To understand the meaning of the term domain specific language or more precisely domain specific programming language the term programming language is defined. There exists no definition which all authors agree upon. Watts therefore proposes [18] some criteria which have to be fulfilled by a programming language:

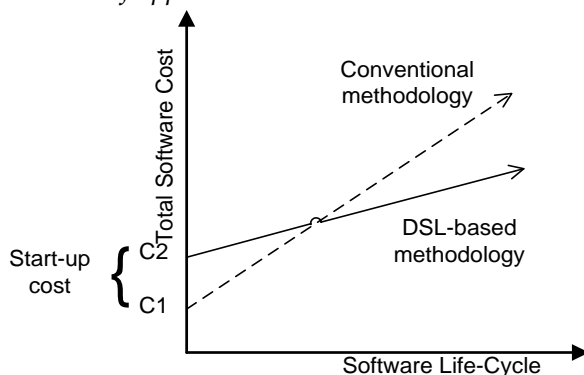
- Must be universal (every problem must have a solution that can be programmed in the language, if that problem can be solved at all by computer);
- Must be implementable on a computer;
- Should also be reasonably natural for solving problems, at least problems within its intended application area.

Programming languages in general can be grouped or classified by different criteria. Possible criteria are the purpose, the paradigm, the generation, whether it is imperative or declarative, and domain specific or general purpose. General purpose languages (GPLs) are less specialized and are suited for a wide area of applications

On the other hand the term domain specific means that the language is explicitly tailored to a target domain. Complex constructs and abstraction of the domain are offered within the language increasing its expressiveness in comparison to GPLs. It is possible to express solutions for domain problems with a lesser effort. The higher abstraction and the compactness and therefore better readability and writability enables a larger group of people with less programming knowledge to be productive using the DSL. This leads to productivity gains in general and also to decreased maintenance costs.

Mernik [48] defines domain specific language as:

*“DSLs are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with GPLs in their domain of application”.*



**Figure 2-2** Cost prediction for DSL-based methodologies [21]

DSL offers different advantages. Productivity and maintainability [57] are increased due to an appropriated domain specific notation. DSLs are more suitable for end-user programming. Domain experts are able to understand, validate, modify, and develop within the language (better readability, writability and high abstraction). The gains can be measured quantitatively and qualitatively. Figure 2-2 shows the advantage of DSLs regarding to long term cost.

Because of the concise nature and the domain fitting notation DSLs are up to a certain degree self-documenting. This also facilitates the embodying of domain knowledge which eases reuse [66] and conservation.

Another advantage is the possibility to validate at domain level [15]. While normal GPL compilers do not know about any domain concept beyond the general language constructs, a DSL can be checked for any domain specific constraint. Just as verification, optimization can be done more effectively at the domain level [18].

On the other hand, a DSL has also potential shortcomings. One drawback is the high development effort which is needed for a new language. The language developer needs at least experience in language design and knowledge about the target domain. He has to find fitting abstractions, the right scope and balance between GPL and DSL constructs. Furthermore the language must be implemented and maintained.

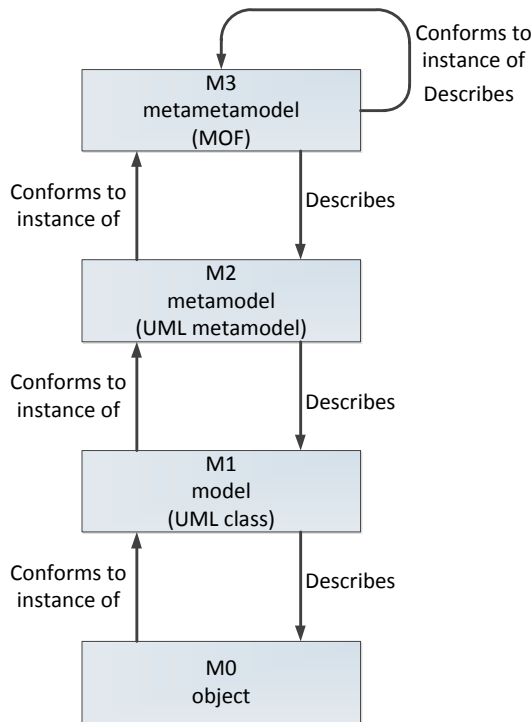
Other problems are tooling, user training costs and performance. While general purpose languages have a strong tool support, corresponding tools for a new DSL have to be created. Creating a tool for a DSL is a time consuming process which adds to the total costs caused by language design and implementation. Without a development methodology and suitable tools the risk is high that the DSL development costs surpass the estimated saving by using a DSL.

The mentioned training costs originate from the fact that possible DSL users have by definition never used the language before. However this is mitigated as in most cases the new language should match the domain expert's expectations.

Often a DSL will suffer from a lower performance than a hand written software. As long as performance is not critical the other DSL benefits will make this a minor problem. Nevertheless in some cases performance can be equal or faster because optimization is possible on a high abstraction level but in most cases the potential is limited.

### 2.3.3 Meta-Object Facility (MOF)

As mention before different metametamodels exist. A well-known one is described in the Object Management Group (OMG) Meta Object Facility (MOF) standard [54]. The idea behind MOF is a four-layered architecture as depicted in Figure 2-3.



**Figure 2-3** Example of MOF layers

Sometimes it is referred to as a closed metamodeling architecture because the defined metametamodel conforms to itself. Currently with version 2.0 different flavors of the MOF exist: the Complete MOF (CMOF), the Essential MOF (EMOF) and the Semantic MOF. CMOF is the whole MOF whereas EMOF is only a subset of the most important elements.



The widespread use of the Universal Modeling Language (UML) is one of the reasons for the recent popularity of MOF. The current version is the base for UML2. All different diagram types are defined conforming to the metamodel. The MOF is the foundation for OMG Model Driven Architecture (MDA), too.

### **2.3.4 Model Driven Architecture (MDA)**

The Model-Driven Architecture (MDA) [47] is an approach to IT systems specification defined by the OMG. MDA consists of a set of guidelines which support MDE. These guidelines mainly concern the structure of software specifications in the terms of models and their transformations. OMG has also defined a standard language for model transformation called Query View Transformation (QVT) [54].

MDA separates the system functionality specification and its further implementation on a specific platform. MDA distinguishes three abstraction levels of models:

- Computational Independent Model (CIM). A model of a system that is focused on the environment of the system and on the specific requirements of the system. Represents the computational independent viewpoint, and hides the structural details and, of course, the details related to the targeted platform.
- Platform Independent Model (PIM). A model of a system that is independent of the platform or technology information that is used to implement it.
- Platform Specific Model (PSM). A model of a system that includes the specific technology information to be used for implementation on a specific platform.
- Code Model (CM). Since the PSM is a platform, the code generation can be automating.

Through such separation of the specification from the implementation technology platform, the specification can be reused for different technologies and multiple platforms. This provides flexibility on changes in the implementation technologies or adopting the system functionality to the current standards. Different applications can be integrated with prepared models through applying mappings to specific platforms.

MDA is based on four principles [5]:

- The models are expressed in a well-defined notation that is the key to understanding the system-level solutions;
- The construction of the systems can be organized around a set of models by imposing a series of transformations between them, organized in a framework of layers and transformations;
- It supports the description of models in a set of meta-models that facilitates the integration and transformation between models, and is the basis for automation through tools.
- The adoption and acceptance of this modeling approach requires industry standards to provide access to users and promote competition among providers.

## **2.4. Technological Areas**

At this point we present the technological areas related to the thesis.

### **2.4.1 Eclipse**

The Eclipse community [65] is an open source community whose projects are focused on creating an open development platform comprised of extensible frameworks, tools for creating, deploying and managing software across the lifecycle.

One of his most important projects is Eclipse. Eclipse is an open source project, robust, with multiple features, commercial-quality industry platform for the development of highly integrated tools. Integrated Development Environment (IDE) uses different modules to enhance its functionality; this is an advantage over other monolithic environments where functionality is not configurable.

In short, the nature of this tool makes it an open, extensible IDE for multiple purposes. This work will be of particular interest the Eclipse Modeling Framework (EMF), a framework for managing models and code generation from models described in XMI. EMF is described in detail in the next subsection.

### **2.4.2 EMF**

The EMF project is a modeling framework and code generation facility for building tools and other applications based on structured data models. It starts with a model specification described in XMI. EMF provides tools to produce a set of Java classes for the model.

You can generate a set of adapter classes that enable views and edition commands based on the model.

Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose through which can be imported to EMF. The most important thing is that EMF provides the foundation for establishing interoperability with other tools and applications based on EMF. With regard to the relationship of EMF to OMG and MOF, EMF started out as an implementation of the MOF specification matured from the experience gained in the development of tools by the developers of Eclipse. EMF can be seen as an efficient implementation of joint use of the MOF API. However, to avoid confusion, the EMF metamodel based on the MOF core is called Ecore [65].

# Chapter 3

## Perspectives for Multi-agent Systems

The importance of the requirements phase in software development is widely known. However, capturing and modeling the requirements of a system is not a trivial task. Moreover, a recent study on the application of requirements engineering techniques in the development of a multi-agent system [7] found that 79% of the current methodologies for MAS development use requirements engineering techniques which have been adapted from other paradigms (object orientation, knowledge engineering, etc.). However, these techniques and notations may not be sufficient to cover the nature of MAS. In particular, MAS require abstractions, techniques, and notations that have been specifically tailored to this domain and to cover characteristics that are not normally necessary in conventional software systems such as pro-activity, adaptability, collaboration, truth, or disposition.

To capture common and special characteristics of a MAS we propose four basic perspectives for the modeling of MAS requirements: functional, structural, organizational, and social behavior. In order to contextualize these perspectives, an overview of them is presented in this chapter which emphasizes both organizational and social behavior aspects, since these are key aspects for the development of MASs.

### 3.1. Functional Perspective

The functional perspective shows the semantics associated with the organizational roles' services that are motivated by the occurrence of events. In this context, we understand an organizational role to be the representation of an abstract entity that provides (multiple) system methods or services. An event is something that occurs in the environment and to which the organizational role reacts by running a method or service. This perspective focuses to model the functional requirements to be met by the roles in the future MAS.

### 3.2. Structural Perspective

The structural perspective shows the system architecture in terms of entities and the static relationship between them. The modeling of these entities and relationships provides an abstract structural view of the system. We believe that this view is necessary to identify the entities that will be needed to build the future MAS. If the static and structural relationships are to be captured accurately, the development method must include formalisms and techniques to specify relationships of hierarchy (inheritance), semantic dependency (association) and part-of relations (aggregation).

### 3.3. Organizational Perspective

In the organizational perspective, the organization is represented as a system that has certain goals. The organization attains these goals through consistent actions, which use system resources and alter the desired system state [24]. To get a clearer idea of this perspective we present two important definitions in the literature. Our work is based on the second.

In the work presented in [25] an organization is seen as a collection of agents that can be considered together in groups, playing roles or regulated by organizational rules. The agents can interact only inside a group in which they play roles. An agent can play one or many roles and enter into one or many groups. A role is a general concept to which a MAS architect can associate various semantics. They define the followings features of organizations:

- An organization is constituted of agents that manifest a behavior;
- The overall organization may be partitioned into suborganizations;
- Behaviors of agents are functionally related to the overall organization activity (concept of role);
- Agents are engaged into dynamic relationships, which may be “typed” using taxonomy of roles, tasks and protocols;
- Types of behavior are related through relationships between roles, tasks and protocols.

An important element of organizations is the concept of role. A role is a description of an abstract behavior of agents. A role describes the constraints (obligations, requirements, skills) that an

agent will have to satisfy to obtain a role, the benefits (abilities, authorization, profits) that an agent will receive in playing that role, and the responsibilities associated to that role. A role is also the placeholder for the description of patterns of interactions in which an agent playing that role will have to perform.

Another important definition is presented in [61]. They consider that the human organizational metaphor is very adequate for systems which are situated in open and changing environments, in which:

- A software system is conceived as the computational instantiation of a (possibly open) group of interacting and autonomous individuals (agents);
- Each agent can be seen as playing one or more specific roles: it has a well-defined set of responsibilities or subgoals in the context of the overall system and is responsible for pursuing these autonomously. Such subgoals may be both altruistic (to contribute to a global application goal) or opportunistic (for an agent to pursue its own interests);
- Interactions are no longer merely an expression of interdependencies, and are rather seen as a means for an agent to accomplish its role in the system. Therefore, interactions are clearly identified and localized in the definition of the role itself, and they help characterize the overall structure of the organization and the position of the agent in it;
- The evolution of the activities in the organization, deriving from the autonomous execution of agents and from their interactions, determines the achievement of the application goal, whether an a priori identified global goal (as, e.g., in a workflow management systems where altruistic agents contribute to the achievement of a specific cooperative project), or a goal related to the satisfaction of individual goals (as, for example, in agent-mediated auctions, whose purpose is to satisfy the needs of buyer and seller agents), or both (as, for example, in network enterprises exploiting market mechanisms to improve efficiency).

Some simpler systems can be viewed as a single organization, as soon as the complexity increases modularity and encapsulation principles suggest dividing the system into different suborganizations, with a subset of the agents being possibly involved

in multiple organizations. In each organization, an agent can play one or more roles, to accomplish which agents typically need to interact with each other to exchange knowledge and coordinate their activities. These interactions occur according to patterns and protocols dictated by the nature of the role itself (i.e., they are institutionalized by the definition of the role). In addition, the MAS is typically immersed in an environment (i.e., an ensemble of resources) that the agents may need to interact with to accomplish their role. Interactions with the environment occur via some sorts of sensors and effectors (i.e., mechanisms enabling agents to perceive and act upon some part of the environment). That portion of the environment that agents can sense and effect is determined by the agent's specific role, as well as by its current status.

### 3.4. Social Behavior Perspective

The social behavior perspective shows the possible sequences of events or services to which an agent can respond or that occur in its lifetime, along with interaction aspects such as communication between agents, and this is often represented as state or activity diagrams. As is discussed above, in addition to organizational, structural, and functional properties, a MAS also requires characteristics that are not normally required in conventional software systems, such as pro-activity, adaptability, collaboration, truth, or disposition. These characteristics are denominated as social behavior. We therefore believe that covering this perspective in a proposal for modeling requirements for MAS is an important contribution towards the development of such systems, since the essence of these systems is the performance of complex tasks that other types of systems are not capable of solving.

#### 3.4.1 Classification

In order to properly structure and organize the features of social behavior requirements, we briefly present the classification scheme of agent characteristics defined in [32]. According to the authors, three main attributes of an agent are defined: (i) *autonomy*, which refers to the fact that an agent should run independently, with little or no human intervention, (ii) *temporal continuity*, which signifies that an agent should run continuously rather than simply perform a task and finish, and (iii) *social skills*, which signifies that an agent should

possess some form of social skills, since the agent's advantages lie in its interactive communication with other agents. In addition to these core attributes, an agent can also be classified according to the following social behavior characteristics:

- *Pro-activeness*: this refers to how the agent reacts to -and reasons about - its environment, and how it pursues its goals. The agent can directly react to stimuli in its environment by mapping an input from its sensors directly to an action, or it can take a purely planning, or goal-oriented, approach to achieve its goals. This last approach relies upon utilizing planning techniques.
- *Adaptability*: this describes an agent's ability to modify its behavior over time. In fact, the term "agent" is often taken to implicitly mean "intelligent agents", which combine traditional artificial intelligence techniques to assist in the process of autonomously performing tasks. This feature includes other sub-features such as learning and sub-submission.
- *Mobility*: this refers to the agents' capability of transporting their execution between machines on a network. This form of moving can be physical, where the agent travels between machines on a network, or logical, where an agent which is running on a single machine is remotely accessed from other locations on the Internet.
- *Collaboration*: collaboration among agents underpins the success of an operation or action in a timely manner. This can be achieved by being able to coordinate with other agents by sending and receiving messages using some form of agent communication language, and permits a high degree of collaboration, thus making social activities such as distributed problem solving and negotiation possible. Moreover, it is possible for agents to collaborate without actual communication taking place. The interaction of agents with resources and their environment may lead to the emergence of collaborative or competitive behavior.
- *Veracity*: this refers to the agent's ability to deceive other agents via their messages or behavior. An agent can thus be truthful in failing to intentionally deceive other players. Moreover, an agent that is untruthful may try to deceive other agents, either by providing false information or by acting in a misleading way.
- *Disposition*: this refers to the agent's "attitude" towards other agents, and its willingness to cooperate with them. An agent may always attempt to perform a task when asked to do so



(benevolent), or may act in its own interests to collaborate with other agents only when it is convenient to do (self-interested), or it might try to harm other agents or destroy them in some way (malevolent).

The above characteristics in the classification represent to some extent abstraction of human social behavior, and are those that differentiate agent paradigms from traditional software development. In this work, we use this classification to study the characteristics of social behavior and to propose mechanisms for the definition and specification of requirements of these types. In particular, and as a starting point, in this work we will focus on the following characteristics: proactiveness, collaboration, veracity, and disposition. Other characteristics such as adaptability or mobility will be considered in future work.

Social behavior is a skill that must have an agent in a MAS. Moreover, if we consider the organizational metaphor, an agent can, at different times in its life-cycle, play one or more specific roles, which in turn have a set of responsibilities and goals. We therefore propose to identify these features of social behavior in the requirements modeling process at role level, through an analysis of the goals that need to be attained. Therefore, in the later phases of the software development, when an agent has to be defined, the corresponding roles of which a given agent will be composed will determine the agent's complete social behavior.

# Chapter 4

## Related Work

This chapter presents the result of a literature review with which to study the existing works in the interrelated field of RE, Agent-Oriented approaches, and MDSD. In particular, it presents the most relevant agent-oriented methodologies found in the literature, approaches of agent-oriented methodologies in the MDSD context, proposals of requirements modeling for MAS, and finally, the results of a systematic review of the use of RE in the development of MAS.

### 4.1. Agent-Oriented Methodologies

Several complete methodologies for the analysis and design of MASs have been proposed so far. We present in this section, some of the most important proposals with the purpose of: analyze the approach adopted, the general elements identified, the phases of software development contemplated, and other features.

The Gaia methodology [73] addresses the analysis and design of agent-based systems without directly dealing with: a particular modeling technique, implementation issues, or the activities of the requirements capturing and modeling, and specifically of early requirements engineering. It considers the system as a society or organization. The organization consists of a collection of roles, that have relationships with one another, and that take part in interactions with other roles. Each role is defined by four attributes: responsibilities (its functionality, described as liveness and safety properties), permissions (in terms of rights, identify the resources that are available to the role, such as information resources), activities (those computations that can be performed without interacting with others), and protocols (the way that it can interact with other roles). These protocols are further defined in the interaction model. Design in Gaia produces two models: an agent model, which identifies agent types (essentially, as aggregation of roles) and their instances in a system, and a services model (functions of each agent). The Gaia process starts with the analysis phase, whose aim is to collect and organize the specification which is the basis for the design of the computational organization. The output of the analysis phase is

exploited by the design phase, which can be logically decomposed into an architectural design phase and a detailed design phase. Once the overall architecture of the system is identified, the detailed design phase can begin.

The MASE (Multi-agent systems Software Engineering) methodology [20] provides guidelines for developing MASs based on a multistep process, most of them supported by the agentTool system [19]. MaSE adopts the object-oriented paradigm (UML), by considering agents as specialized proactive objects that coordinate by means of conversations. In analysis, the requirements are used to define use-cases and application goals and subgoals, followed by use case analysis and the definition of the corresponding sequence diagrams. From these diagrams, it is possible to derive roles and their associated tasks. Roles in MaSE form the foundation for agent class definition, represent system goals during the design phase, have to be played by the agents, and have their interactions. The design phase in MaSE produces an agent class diagram, by assigning roles to specific agent classes, the conversations between agents, the design of internal agent architectures, and the deployment of agents in a system, leading to a complete architecture of the system. A conversation is a coordination protocol between two agents, and it is described with two finite state machines, one for each party (initiator and responder). As explicitly acknowledged by the MaSE designers, the methodology is suitable only for closed agent systems.

MESSAGE [12] is a proposal to integrate different methodologies. It builds on five viewpoints, and is refined and extended with INGENIAS [31]. INGENIAS takes into account the analysis, design, and implementation stages of the software development life cycle. The development of a MAS consists of identifying elements for each viewpoint and then performing a set of activities that are defined in the context of the Unified Process [35]. The agent viewpoint describes an agent's responsibilities with tasks and roles. It also takes into account the control of the agent and defines its goals and the mental states required during execution. The organization viewpoint determines the architecture of a system. Structural relationships are not restricted to hierarchies between roles. These structures are delegated to specialized entities called groups. In the organization model there are also power relationships

among groups, organizations, and agents. The functionality of the organization is expressed using workflows. The environment viewpoint defines the sensors and effectors of the agents. It also identifies available resources as well as already existing agents and applications. The tasks and goals viewpoint main purpose is to justify the execution of tasks in terms of the goals. It also provides the breakdown of tasks and goals. This viewpoint also provides low-level details of tasks in the system, and describes which resources are needed during an execution, which software modules are used throughout the process, and which are the inputs and outputs. Finally, the interaction viewpoint describes how coordination among agents takes place. It goes a step further than UML sequence diagrams since it reflects the motivation of the interaction and its participants. It also includes information about the mental state required in each agent throughout the interaction as well as tasks executed in the process.

Tropos [29] is based on two key features. First, the notion of agent and related mentalistic notions are used in all software development phases, from the early requirements analysis down to the actual implementation. Second, the methodology emphasizes early requirements analysis, the phase that precedes the prescriptive requirements specification. Tropos adopts i\* model [72] which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modeling an application during early requirements analysis. Tropos is intended to support four phases of software development: early requirements analysis, concerned with the understanding of a problem by studying its organizational setting; late requirements analysis, where the system-to-be is described within its operational environment, along with relevant functions and qualities; architectural design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control, and other dependencies; and detailed design, where the behavior of each component is defined in further detail.

MASSIVE - Multi Agent SystemS Iterative View Engineering [45] is a pragmatic method for the design and construction of MAS. MASSIVE is based on a view-oriented approach: different phases can be executed focusing on different aspects of the systems. In the environment view, the environment of the target system is analyzed

from the developer's perspective as well as from the systems perspective. These two perspectives usually differ as the developer has global knowledge whereas the system has only local knowledge. In the Task view, the functional aspects of the target system are analyzed and a task hierarchy is generated that is then used to determine the basic problem solving capabilities of the entities in the final system. Furthermore, the nonfunctional requirements of the target system are defined and quantified as far as possible. The Role view determines the functional aggregation of the basic problem solving capabilities according to the physical constraints of the target system. A role is an abstraction that links the domain dependent part of the application to the agent technology that solves the problem under consideration. In the interaction view, the MAS is considered as an ensemble of interacting agents, in which various forms of competition and cooperation, as well as non-traditional forms of cooperation, may be identified. In the society view, the multiagent systems is considered as a structured collection of agents, organized according a particular organizational model. The Architecture view is a projection of the target system onto the fundamental structural attributes with respect to the system design. The major aspects that are dealt with in this view are the system architecture as a whole and the agent architecture. The System view, finally, deals with systems aspects that affect several of the other views or even the system as a whole. Overall, the MASSIVE process covers analysis, design, implementation, verification and testing, and deployment. No attention is paid to requirements elicitation.

AUML (Agent Unified Modeling Language) [4] is a graphical modeling language that is being standardized by the Foundation for Intelligent Physical Agents (FIPA) Modeling Technical Committee (Modeling TC). It is important to explain that AUML is only a modeling language, not a methodology, and, consequently, lacks the definition of predefined steps for designing the diagrams. AUML builds on the acknowledged success of UML in supporting industrial-strength software engineering, evolving it into the agent's field. The vision of an agent is presented as the next step from the concept of object. AUML recaps the available interest to agent-oriented development methodologies with the acceptance of UML. UML provides an insufficient basis for modeling agents and agent-based

systems. Basically, this is due to two reasons. Firstly, compared to objects, agents are active because they can take the initiative and have control over whether and how they process external requests. Secondly, agents do not only act in isolation but in cooperation or coordination with other agents. Multiagent systems are social communities of interdependent members that act individually. AUML currently presents a set of extensions to UML: the specification of agent interaction protocols, and the representation of social and organizational structures among agents.

DESIRE (framework for DEsign and Specification of Interacting REasoning components) [11] allows the system designer to explicitly and precisely specify both the intra-agent functionality (i.e., the expertise required to perform the domain tasks for which the agent is responsible in terms of the knowledge requirements and the reasoning capabilities) and the inter-agent functionality (i.e., the expertise required to perform and guide coordination, cooperation and other forms of social interaction in terms of the knowledge requirements and the reasoning capabilities). DESIRE views both the individual agents and the overall system as a compositional architecture - hence all functionality is designed as a series of interacting, task-based, hierarchically structured components. Tasks are characterized in terms of their inputs, their outputs and their relationship to other tasks. Interaction and coordination between components, between components and the external world, and between components and users is specified in terms of information exchange, sequencing information and control dependencies. The components themselves can be of any complexity (from simple functions and procedures up to whole knowledge-based systems) and can perform any domain function (e.g. numerical calculations, information retrieval, optimization, et cetera).

## **4.2. Agent-oriented Methodologies in the Context of Model Driven Development**

In this section we discuss some related contributions with regard to MAS modeling in the context of model-driven development approaches with the purpose of: analyze the main concepts modeled, the existence of proposals for model transformation, and other features.

Gaia [73], [5] explicitly models the social aspects of open agent systems, with particular attention on the social goals, social tasks and organizational rules. The methodology is focused on the organizational structure of the system. The main concept of Gaia is the *Agent*, which is part of an *Organization*, collaborates with other *Agents*, provides *Services* and plays several *Roles* that act in a *Communication*, specifying a *Protocol*. The Gaia2JadeProcess [49] shows how systems designed by following the GAIA methodology, and its corresponding models, can be converted to JADE for deployment.

PASSI [16] considers three different domains. The *ProblemDomain* contains concepts such as *Resource*, *Non-Functional Aspects* and *Requirements* that are connected with the *Agent*. *Requirements* are represented with conventional use case diagrams. The *AgencyDomain* covers aspect such as the *Agent*, which has a set of *Roles* that provide a *Service* and solve *Tasks* that includes a set of *Actions*. The *Role* is also connected to *Communication*, which works on *Agent Interaction Protocols* with a set of *Performatives*. The *ImplementationDomain* covers the *FIPA-Platform Agent*, *Service Description*, and *FIPA-Platform Task* concepts. *Collaboration* features are detected at the requirements level. ADELFE [59] specifies a methodology to develop adaptive MAS by concentrating on cooperative behavior. The main concept of ADELFE is the *CooperativeAgent* which has *Skills*, *Aptitudes*, *Characteristics*, and *Communications*. Furthermore, the agent observes *CooperationRules*. An agent also possesses world representations, which are *beliefs* concerning other agents and the physical environment.

INGENIAS [31] recognizes five metamodels that describe MAS views. The *AgentModel* describes single agents, their *tasks*, *goals*, initial *mental state*, and *roles* played. The *InteractionModel* describes how *interaction* among agents takes place. The *TasksandGoalsModel* describes relationships among *goals* and *tasks*, *goal structures*, and *task structures*. The *OrganizationModel* describes how system components (*agents*, *roles*, *resources*, and *applications*) are grouped together, which *tasks* are executed in common, which *goals* they share, and what *constraints* exists in the interaction among agents. Finally, the *EnvironmentModel* defines the *agent's perception* in terms of existing elements of the system. An update of INGENIAS, presented in [57],

introduces the INGENIAS Development Kit (IDK) as a means to provide MDD tools for MAS development. This proposal provides information on pro-activity and collaboration from the analysis level.

In Tropos [29], systems are modeled as *social structures* that contain collections of *actors*, and a set of *dependencies* between them. These dependencies are defined in the *Early Requirements Analysis*. In the *Late Requirements Analysis*, the system is represented as *actors* that have *dependencies* with other *actors* and the *organization*. At the *design level*, features are identified in more detail, such as the *skills* needed by an *actor* to fulfill *goals* and *plans*. Tropos thus models the *pro-activity* and *collaboration* with *skills* and *plans*, but only from the design phase. Tropos methodology uses a modeling language named *i\** [72]. The proposal in [58] addresses the problem of linking requirements analysis to detailed design and implementation in the Tropos, and particularly focuses on building a Tropos metamodel as a Platform Independent Model for an MAS and on how to automatically transform it into a JADE metamodel as the Platform Specific Model.

### 4.3. Modeling requirements for multi-agent systems

The importance of the requirements phase in software development is widely known. However, capturing and modeling the requirements of a system is not a trivial task. In particular, MAS require abstractions, techniques, and notations that have been specifically tailored to this domain. We consider the four basic perspectives for modeling MAS requirements presented in the previous chapter: functional, structural, organizational, and social behavior. This section, presents some proposals for the acquisition and modeling of requirements that cover these four perspectives of a MAS.

The organizational perspective is supported in proposals such as GBRAM [1]. GBRAM is a relevant traditional goal-oriented requirements engineering proposal. It provides a procedural guide for the identification and development of goals and introduces techniques that assist in their methodical and systematic analysis. GBRAM has a great deficiency in terms of formality. This includes the lack of models, formal notations and tools that support the modeling that the method uses. Nevertheless, the guidelines and the



level of clarity it offers are very good. Moreover, GBRAM also emphasizes the verification of the requirements through its refinement stage which specifies certain guidelines to follow, thus making this process more reliable. Therefore it is possible to track the requirements captured, and this is reflected in the traceability offered by the method.

Another proposal for requirements modeling that supports the organizational perspective is the  $i^*$  framework [72]. This framework has been established as the basis for the Tropos methodology [29]. Tropos has been appropriately adapted to the acquisition and modeling of the actors in the system and its environment, i.e., the actors, goals, tasks, interactions, dependencies, resources needed, etc. However, it does not permit a full representation of constraints nor does it propose a modeling environment. Since we consider goal orientation to be of particular interest in the capturing of requirements for MAS, we believe that it is necessary to analyze other methods which are complementary to this approach.

The structural perspective is supported by proposals such as AUML [52]. AUML tends to be asserted as a notational standard in various methodologies; one of the most common proposals for the requirements phase is the adoption of Use Case diagrams. This formalism has shown good results for the representation of functional requirements and is also a good tool for communication with stakeholders. Nevertheless, Use Cases have limitations in capturing qualitative aspects of the environment and interactions with it. In addition, an interesting contribution of AUML is the Agents Interaction Protocol (AIP), which constitutes a central aspect for MAS, specified by means of protocol diagrams.

Another proposal that covers the structural perspective is KAOS [68], a proposal for modeling requirements through goals. KAOS consists of a specification language, a method of elaboration, and the meta-level knowledge which is used as a guide. A KAOS model contains goals, information system requirements, expectations about the system environment, conflicts between goals, obstacles, entities, agents, etc. One of the strengths of the proposal is that of its use of formality to achieve correction. Moreover, the idea of constraint is useful in identifying some of the external problems of

integrity, and this contributes to the robustness of the system. However, the successful implementation of the method depends heavily on the developer's experience in the domain and how well defined the problem to be solved is [33].

Other proposals do not support the organizational and structural perspective. This is the case of CREWS [46], which focuses on the perspectives of functional and social behavior. CREWS is based on system object models that are abstractions of the key features of the different qualities of the problem domain. CREWS uses these models to generate normal course scenarios, and it then uses the theoretical and empirical research into cognitive science, human-computer interaction, collaboration systems and software engineering as a basis to generate alternative courses for these scenarios. A potential weakness of the CREWS approach is that the generation of scenarios is domain-oriented, in contrast with the goal-oriented scenario analysis and the task-oriented Use Case modeling. If the scenarios are intended to validate the requirements, these requirements should be oriented towards the generation of scenarios.

In summary, the organizational perspective is covered by proposals such as GBRAM and  $i^*$ , and the structural perspective is covered in proposals such as KAOS and AUML. Most of the proposals presented in some way cover, either totally or partially, the functional and social behavior perspective, as in the case of CREWS.

#### **4.4. A Systematic Review of Requirements Engineering in the Multi-Agent Systems**

In [7] the authors present a systematic review to determine the current research activity in RE for MAS development. This section briefly presents the most important results obtained in the review, and on which we base and motivate our work.

One of the more important research gap identified by the review is that almost every requirement method proposed for MAS development is adapted from other fields, and very few of them proposed a new method to deal with requirements. Also, few of MAS methodologies are focused on eliciting requirements, while most are focused on modeling and analyzing them. Regarding the concepts and notations, there are several works that use goals, NFR, scenarios,

and role models. In addition, there is interest in the MAS field for the use of patterns to model from an abstract level to a more detailed and architectural design level. Additionally, there is a lack of work on traceability among artifacts produced along the MAS development. Better mechanisms and tools to deal with (pre and post) traceability in the MAS methodologies could help to meet user needs, improve their understanding of the system, and improve the overall quality of the developed software.

## 4.5. Discussion

From a general interpretation of the literature review in the existing works of the interrelated field of RE, Agent-Oriented approaches, and MDSD, the conclusions that influence our work are the following:

- Most of the agent-oriented methodologies mainly focus on the design and implementation phases and do not provide mechanisms for capturing, defining, and specifying software requirements;
- MDSD technology has recently been introduced into AOSE literature and merged with software agent technology to support the development of MAS;
- Considering the perspectives proposed in Chapter 3 to adequately represent a MAS, to the best of our knowledge no requirements modeling methods for MAS completely covers the perspectives needed to adequately represent a MAS: functional, structural, organizational, and social behavior. There is therefore a need to develop RE methods covering all four properties of a MAS;
- The results of the systematic review of [7], motivates our work to propose a new method to deal with MAS requirements;
- Also, the results of the systematic review of [7], guides us regarding the concepts and notations used to identify and model the software systems requirements;
- Finally, lack of work on traceability among artifacts produced along the MAS development identified in the systematic review of [7] influence in the decision to support the traceability including our work in a MDSD context.

# Chapter 5

## Modeling Requirements for Multi-Agent Systems

The main objective of this thesis is to discuss and propose an approach for the definition and specification of requirements for MAS. This approach is being developed as a generic method of requirements modeling for MAS and will provide a well-defined framework not only for dealing with structural and functional aspects of the systems, but also for dealing explicitly with organizational aspects of the system and different types of social behavior aspects in earlier phases of software development.

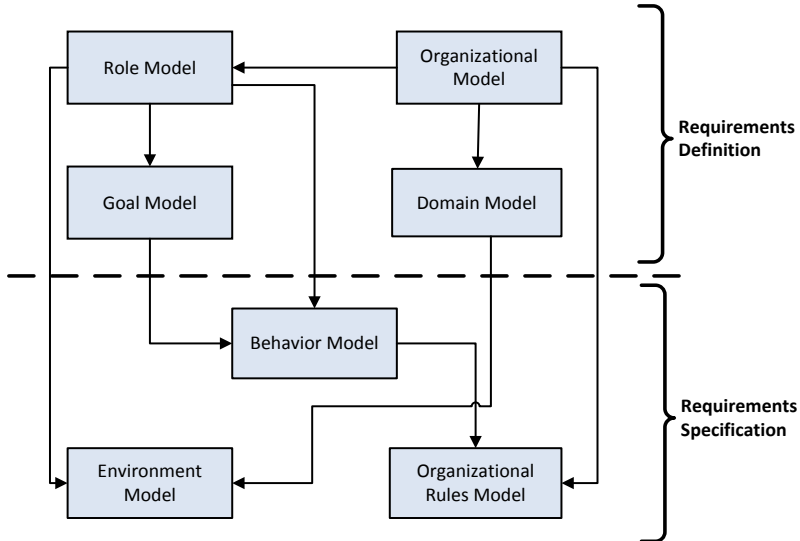
This chapter presents the requirements modeling proposal to support the definition and specification of MAS requirements, covering the functional, structural, organizational, and social behavior perspectives of a MAS.

### 5.1. Proposal

The proposal is decomposed into two main phases: *Requirements Definition* and *Requirements Specification*. The user's specific needs are identified in the *Requirements Definition* phase. In particular it is identified: the organizational structure of the system; the roles that are required in each sub-organization; the roles goals; and relevant entities of the environment. The detailed requirements for developers are specified in the *Requirements Specification* phase. The specifications extracted from the *Requirements Definition* activity are refined, and the level of detail increased, in order to identify artifacts which are closer to the analysis and development of the system: the social behavior needed for the roles to carry out the role goals; activities and interactions; resources of the system; the permissions that roles have in those resources and organizational rules.

Moreover, the process is based on the definition of models needed to describe the problem in more concrete aspects that form the different perspectives of the system. Figure 5-1 shows an

overview of these models and their respective relations. In the following two sections we describe in detail the models to be identified in each of the phases proposed.



**Figure 5-1** Models of the proposal and its relationships

### 5.1.1 Requirements Definition

The objective of the *Requirements Definition* phase is to acquire and represent the software requirements. This phase starts with the definition of the Organizational Model, partially the Role Model, and the Goal Model (in the order listed). The information obtained when defining this three models will serve to complete the Role Model and to define the Domain Model. Finally, the output of this phase is used as input for the construction of the *Requirements Specification* phase models.

The Organizational Model establishes the Mission Statement, which is the global goal of the organization, and determines the sub-organizations that compose the global organization. The Role Model determines the roles involved in each sub-organization, and is used to detect inheritance relations between roles and reasoning about structural relationships. The Goal Model determines the roles goals. Finally, the Domain Model is used to identify entities that could be used as resources of the organization.

In order to obtain a clear view of the models used, each of them is presented as follows.

#### **5.1.1.1 The Organizational Model**

It consists of the Mission Statement, which is defined in natural language, with a recommended extension of one or two paragraphs. The Mission Statement is the root of a hierarchy, which is successively refined to identify the system hierarchy of sub-organizations. A sub-organization is a part of the system that aims to achieve a goal in the system and weakly interacts with other parts of the system (low coupling).

The Organizational Model allows us to identify elements of the organizational perspective through the identification of the Mission Statements and the decomposition of the system into sub-organizations. Since the Mission Statement identifies the overall goal within the organization as a whole, it also provides information about the functional perspectives.

#### **5.1.1.2 The Role Model**

A role is the representation of an abstract entity that has (multiple) system goals. As was previously mentioned, the Role Model describes the roles that belong to the sub-organizations. Each sub-organization is refined into the roles that compose it. The purpose of this model is to represent the different roles found in each sub-organization and to reason about their special relationships. The special relationships between roles can serve to identify the common properties between the roles in order to create a hierarchy of roles using inheritance relationships.

Then, the resulting Role Model comprises the information of the roles needed in each organization to carry out the goals of the same, and information of the inheritance relations between roles.

The Role Model allows us to identify elements of the organizational perspective by identifying the roles that make up the sub-organizations, and elements of the structural perspective by identifying inheritance relations between roles.

### 5.1.1.3 The Goal Model

A goal is a task or a process that involves a series of tasks, which is carried out by a role in the sub-organization. The Goal Model describes the goals that a role must accomplish. Each role is refined into the goals to be met, these being the last level of the hierarchy of mission statement, sub-organization, roles and goals.

The Goal Model allows us to identify elements of the functional perspective by identifying the goals that each role has to perform.

### 5.1.1.4 The Domain Model

Represents the entities identified in the problem domain. The purpose of this is to identify key concepts and relationships, thus representing a first structural view. First, the domain entities relevant to the application domain are identified. An entity can be both a physical entity as a concept. These entities are seen from the point of view of the application domain, and implementation details are therefore avoided at this level.

Associations and inheritance relationships between domain entities are also represented. Finally, we can refine the model with inheritance relationships.

The identification of these domain entities and their relationships allows us to extract information for the structural perspective and to partially extract information for the organizational perspective.

## 5.1.2 Requirements Specification

The objective of the *Requirements Specification* phase is to specify the software requirements base on the information previously identify in the *Requirements Identification* phase. This phase starts with the specification of the Behavior Model, using information identify in the Role Model and the Goal Model of the previous phase. Then, the specification of the Environment Model is constructed based on the information identify in the Role Model and the Domain Model of the previous phase, and finally the Organizational Rules Model is specified based on the Behavior Model of the current phase and the Organizational Model of the previous phase.

The Behavior Model represents the social behavior needed between roles to accomplish their goals, and the decomposition of goals into tasks and protocols in order to understand the internal flow of a role to determine its responsibilities. The Environment Model represents the permissions of the roles identified in the Role Model with regard to the resources of the Domain Model. And finally the Organizational Rules Model represents the constraints of the organization's behavior.

In order to obtain a clear view of the models used, each of them is presented as follows.

#### **5.1.2.1 The Behavior Model**

The first purpose of this model is to reason about the special relationships needed between roles to accomplish their goals. The special relationships between roles can serve to identify social behavior relationships between roles in different sub-organizations. In this proposal, we rely on the classification of social behavior presented in the Chapter 3, and identified three types of social behavior with their respective properties: collaboration (communicative, non-communicative), disposition (benevolent, self-interested, malevolent), and veracity (truthful, untruthful). A social behavior relation between two roles could be of one or more property, since the relation is dynamic, i.e. it may alter depending on the agent that will eventually play the role.

The second purpose of this model is to shows a sequence of steps that represent the flow of activities needed to achieve the goals identified in the system. A representation of the flow of tasks could be useful: to understand the logical flow of a role; to complement the information regarding social behavior identified with the social behavior relations; and to help to identify new information when one role needs to work with others in order to accomplish a task.

The resulting Behavior Model comprises the information represented in: one or more social behavior diagrams as needed for each sub-organization for the social behavior relations between roles, and one or more activities diagrams for each goal.

The information identified with the social behavior diagrams allows us to express elements of the structural, organizational and



social behavior perspectives. The identification of the flow of activities in the activity diagrams allows us to extract information for the functional perspective, and the identification of interactions between different roles in the same diagrams allows us to identify information for the social behavior perspective.

#### **5.1.2.2 The Environment Model**

It represents the permissions of the roles with regard to the entities identified in the Domain Model. For each role identified in the Role Model, resources are established for those who can legitimately access them. Finally the permissions (perceive or modify) are established.

The identification of these permissions offers information of the structural and functional perspectives of the system.

#### **5.1.2.3 The Organizational Rules Model**

It identifies and represents the general rules concerning the organization's behavior. These rules can be viewed as general rules, responsibilities, restrictions, the desired behavior, and the sequence or order in such conduct. These rules will be represented by building on GBRAM, in which two types of dependency relationships between goals are distinguished: precedence and restriction, which are represented by the symbols  $<$  and  $\rightarrow$  respectively, and by adding a relationship to the proposal to represent general rules of the system, which is represented with only natural language. The precedence dependency relationship exists between two goals G1 and G2, if goal G1 must be completed before goal G2 can be achieved. The constraint dependency relationship exists between two goals G1 and G2, if goal G1 is met, then goal G2 must be met too.

This information contributes to extract information for organizational, structural and functional perspectives of the system.

# Chapter 6

## Requirements Metamodel for Multi-Agent Systems

Model-driven software development is a proposal through which to maximize productivity, enhancing aspects such as software reusability, interoperability and improved adaptation of technological change. Model Driven Architecture (MDA) [47] is one of the most popular model-driven software development approaches and is based on the set of standards proposed by the OMG [54]. MDA advocates taking models as the main artifacts of software development as a series of model transformations.

The OMG standard denominated as MOF [55] proposes a modeling architecture based on four layers. In the top level, or M3, resides the meta-metamodel which is used as languages to define modeling languages. In the M2 level reside the metamodels, which are models used to describe models. In the M1 level reside the models, which are instances of level M2 metamodels. Finally, the instances of these models reside in the M0 level.

This chapter presents a software requirements metamodel (M2 level of the MOF architecture) for MAS and the associated graphical syntax. Figure 6-1 shows the software requirements metamodels for MAS. The aim of this metamodel is to define a common set of requirements' core concepts for the development of MAS in the context of MDE approaches, and thus bridging the gap between MAS requirements, captured as requirements models, with analysis and design models by defining the corresponding transformations.

The metamodel represents each model of the proposal: Organizational Model, Role Model, Goal Model, Domain Model, Social Behavior Model, Environment Model, and Organizational Rules Model.

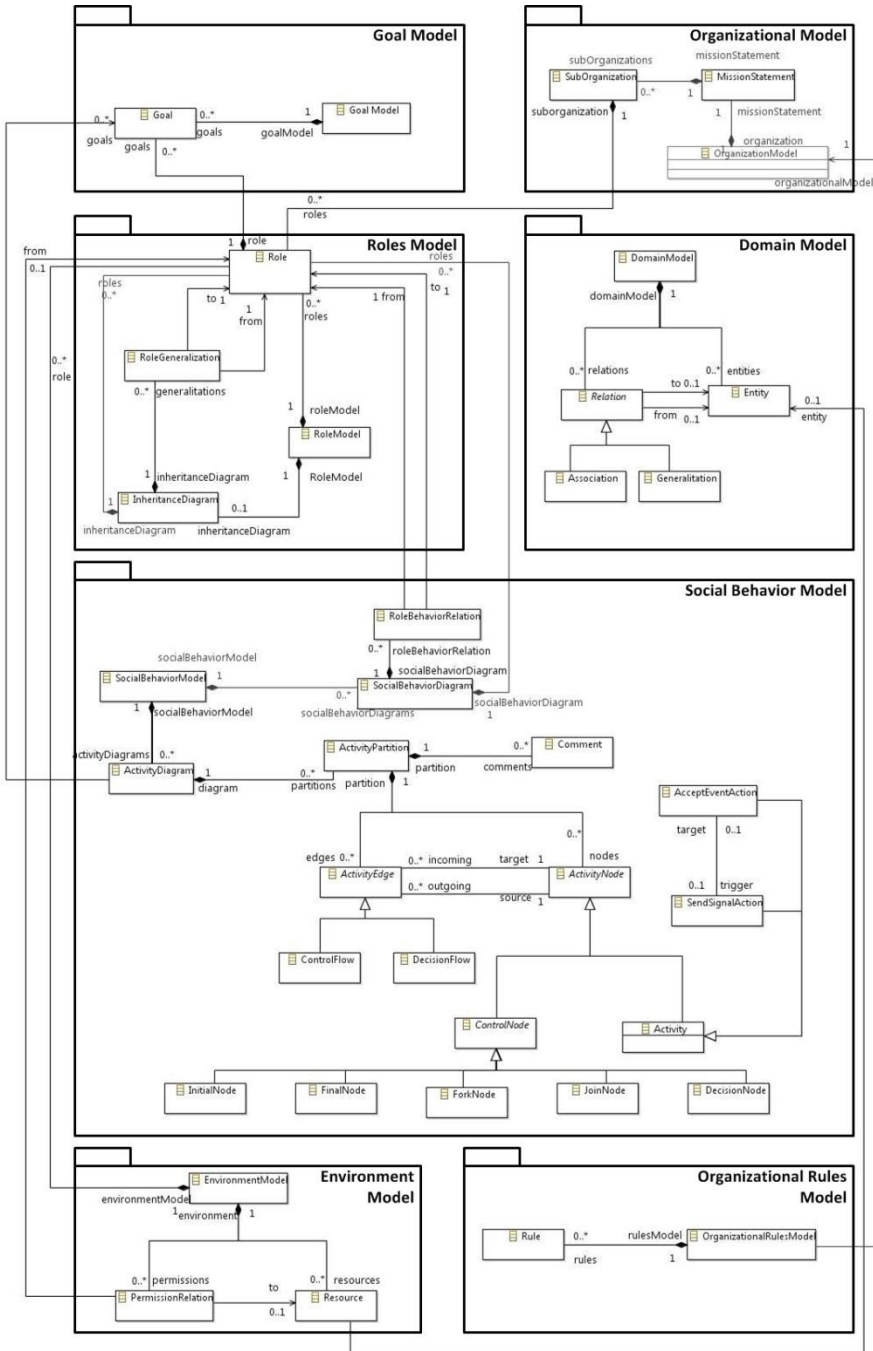


Figure 6-1 Requirements metamodel for MAS

## 6.1. Organizational Model

The Organizational Model allows the specification of the mission statement of the system and the sub-organizations of which the system is composed. A sub-organization is a part of the system that aims to achieve goals in the system and weakly interacts with other parts of the system. Figure 6-2 shows the Organizational Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Role Model, and Organizational Rules Model. In Figure 6-2 we can observe that the *OrganizationalModel* metaclass is a container for the elements of this model. The aggregation relationship between the *OrganizationalModel* and *MissionStatement* metaclasses indicates that an *OrganizationalModel* is composed of these elements. Also, the aggregation relationship between the *MissionStatement* and *SubOrganization* metaclasses indicates that a *MissionStatement* is composed of these elements.

Moreover, the aggregation relationship between the *SubOrganization* and *Role* metaclasses indicates that a *SubOrganization* is composed of these elements. The *Role* metaclass is part of the Role Model. The relationship between the *OrganizationalModel* and the *OrganizationalRulesModel* indicates that an Organizational Model is related to a Organizational Rules Model. The *OrganizationalRulesModel* metaclass is part of the Organizational Rules Model.

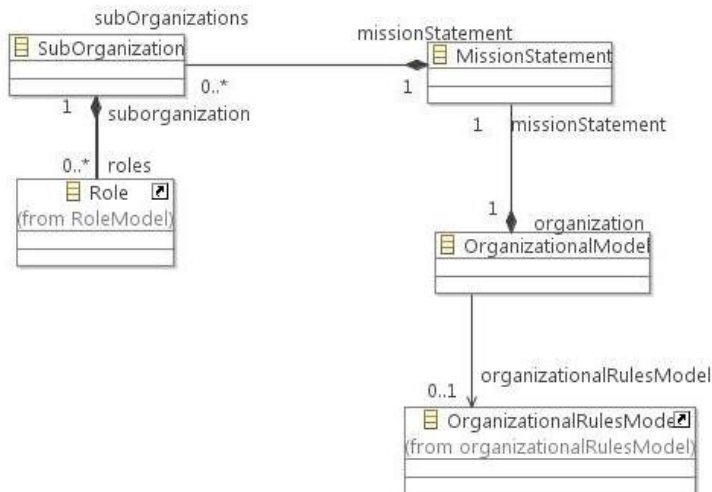
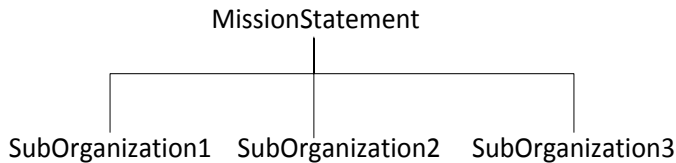


Figure 6-2 Organizational Model

A Refinement Tree is used for the graphical syntax of this model (see Figure 6-3), which is used to represent a hierarchy, based on a decomposition which is independent of the future software structure. The *MissionStatement* is the root of the tree, and following with a decomposition of the *MissionStatement* into a hierarchy of *SubOrganizations*.



**Figure 6-3** Organizational Model Graphical Syntax

## 6.2. Role Model

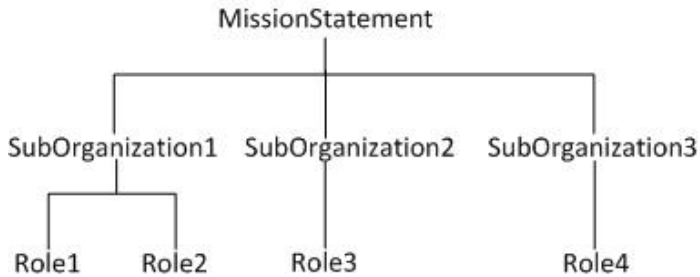
The Role Model, as was previously mentioned, represents the roles involved in each sub-organization. A role is the representation of an abstract entity that has (multiple) system goals in one or more sub-organizations of the system. In addition to all the roles identified, it is also necessary to define the inheritance relations between roles.

Figure 6-4 shows the Role Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Organizational Model, Goal Model, Social Behavior Model, and Environment Model. In Figure 6-4 we can observe that the *RoleModel* metaclass is a container for the elements of the same model. It is composed of roles, and one inheritance diagram represented in the metamodel by the aggregation relationship between the *RoleModel* and the *Role* metaclasses, and the aggregation relationship between the *RoleModel* and the *InheritanceDiagram*. The *InheritanceDiagram* metaclass is composed of a set of roles and a set of generalization relations between roles, represented in the metamodel as an aggregation relationship between the *InheritanceDiagram* and *Role* metaclasses, and the *InheritanceDiagram* and *RoleGeneralization* metaclasses, and as a two association relation between *RoleGeneralization* and *Role* metaclasses, in which the source of the relation is represented with the association labeled *from*, and the target of the relation is represented with the association labeled *to*.

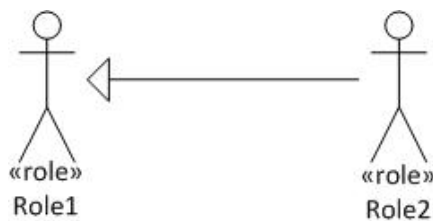


relationship between the *PermissionRelation* and *Role* metaclasses with the label *from* indicates that the role is the source element of the relation.

A Refinement Tree is used for the graphical syntax of this model, to represents the roles that each sub-organization contains (see Figure 6-5). After defining the decomposition of the *MissionStatement* into a hierarchy of *SubOrganizations* we then define the decomposition of each *SubOrganization* into *Roles*. Also, the UML Use Case Diagram is used to represent the inheritance relations between roles (see Figure 6-6). In this diagram the roles are represented as actors which are labeled with the stereotype «role». In addition, the inheritance relations are represented with the corresponding diagram relation.



**Figure 6-5** Role Model Graphical Syntax: Refinement Tree

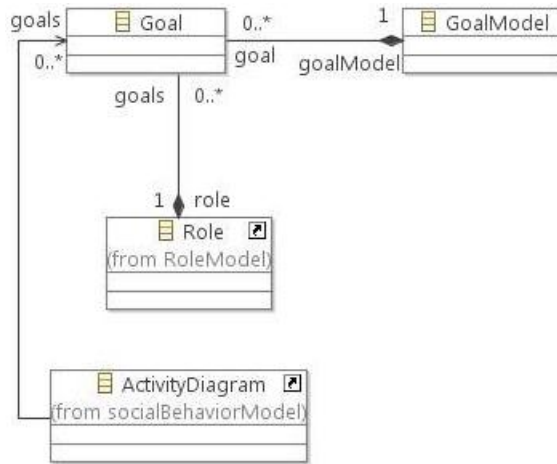


**Figure 6-6** Role Model Graphical Syntax: Inheritance relation diagram

### 6.3. Goal Model

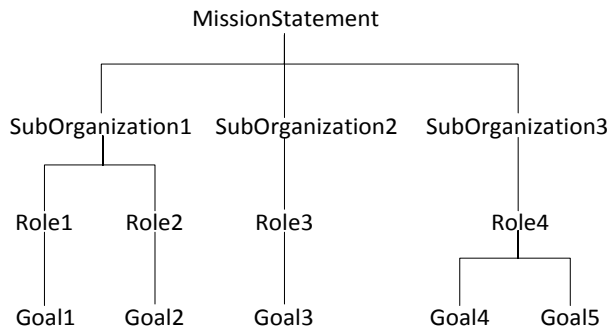
The Goal Model represents the goals associated with each role. Goals are tasks which are carried out by a role in a sub-organization. Figure 6-7 shows the Goal Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Role

Model, and Social Behavior Model. In Figure 6-7 it is possible to observe that the *GoalModel* metaclass is a container for the element of this model. As mentioned previously, a role has a set of goals which are represented in the metamodel as an aggregation relationship between the *Role* metaclass from the Role Model, and the *Goal* metaclass. Moreover, from the Social Behavior Model an *ActivityDiagram* metaclass correspond to zero or more *Goals*, represented with the relationship between the metaclasses.



**Figure 6-7** Goal Model

A Refinement Tree is used for the graphical syntax of this model, to represent the goals that each role has to fulfill (see Figure 6-8). After defining the decomposition of the *SubOrganizations* into a hierarchy of *Roles* we then define the decomposition of each *Role* into *Goals*.

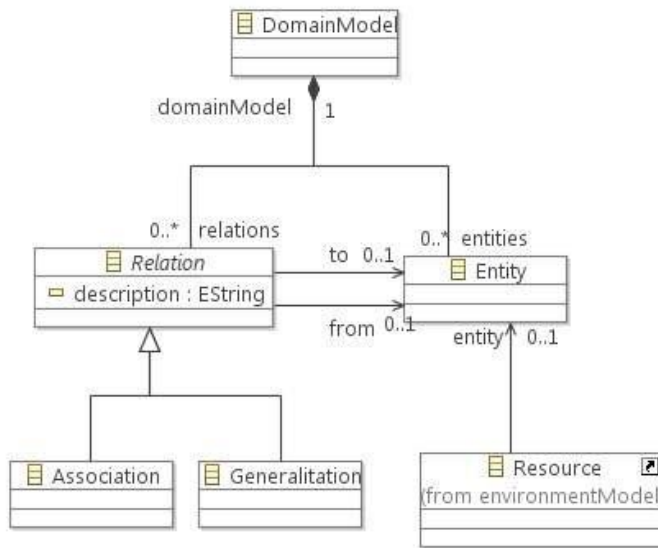


**Figure 6-8** Goal Model Graphical Syntax



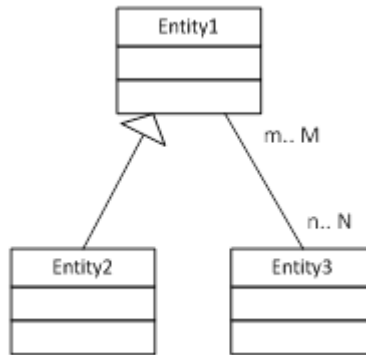
## 6.4. Domain Model

The Domain Model represents the entities identified in the problem domain which could be used as the organization's resources. Associations and inheritance relationships between domain entities are also represented. Figure 6-9 shows the Domain Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Environment Model. In Figure 6-9, the *DomainModel* metaclass is a container for the elements of this model. It is composed of entities and relations, represented in the metamodel with the *Entity* and *Relation* metaclasses. The *Relation* metaclass is an abstract class representing the concept of relation between two entities. A relation can specialize in a generalization or association, represented in the metamodel by the *Generalization* and *Association* metaclasses. The *Association* metaclass can be annotated with multiplicity (source and target): zero, one, many or a constant. The relationship between the *Entity* metaclass and the *Resource* metaclass from the Environment Model indicates that a resource is related to an entity from the Domain Model.



**Figure 6-9** Domain Model

The UML Class Diagram is used for the graphical syntax of this model, to represent the entities and the relations between entities (see Figure 6-10).



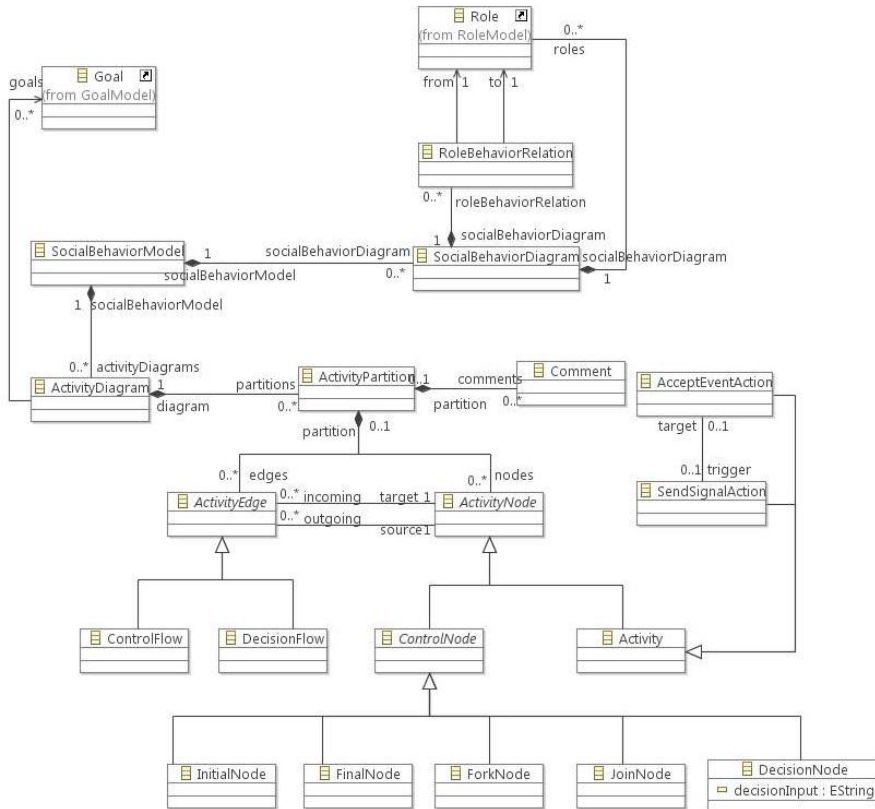
**Figure 6-10** Domain Model Graphical Syntax

## 6.5. Social Behavior Model

The Social Behavior Model represents the decomposition of goals into tasks and protocols in order to understand the internal flow of a role to determine its responsibilities.

Figure 6-11 shows the Social Behavior Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Role Model, and Goal Model. The *SocialBehaviorModel* metaclass is a container for the elements of this model (see Figure 6-11). It is composed of social behavior diagrams and activity diagrams, represented as an aggregation relation between the *SocialBehaviorModel* and *SocialBehaviorDiagram* metaclasses, and *SocialBehaviorModel* and *ActivityDiagram* metaclasses. The *SocialBehaviorDiagram* metaclass is composed of a set of roles and a set of role behavior relations between roles, represented in the metamodel as an aggregation relationship between the *SocialBehaviorDiagram* and *Role* metaclasses from the Role Model, *SocialBehaviorDiagram* and *RoleBehaviorRelation* metaclasses, and as a two association relation between *RoleBehaviorRelation* and *Role* metaclasses from the Role Model, in which the source of the relation is represented with the association labeled *from*, and the target of the relation is represented with the association labeled *to*. On the other hand, the *ActivityDiagram* metaclass is a container of activities. It is composed of different partitions, represented in the metamodel as the *ActivityPartition* metaclass. A partition is, in turn, composed of nodes, edges and comments. This relation is represented as aggregation relations from the *ActivityPartition* metaclass to the *ActivityNode*,

*ActivityEdge*, and *Comment* metaclasses. The *ActivityNode* metaclass is an abstract class which can specialize in an *Activity* or *ControlNode*. Each activity diagram represents the task to be performed to fulfill a goal. This relation is represented as the relation from *ActivityDiagram* metaclass to the *Goal* metaclass from the Goal Model.

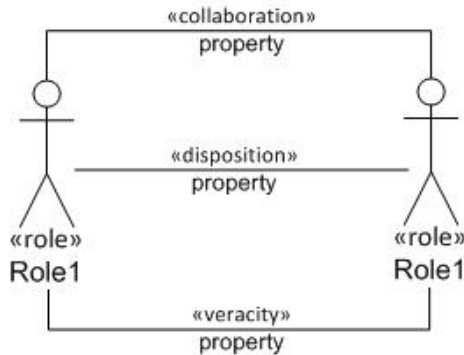


**Figure 6-11** Social Behavior Model

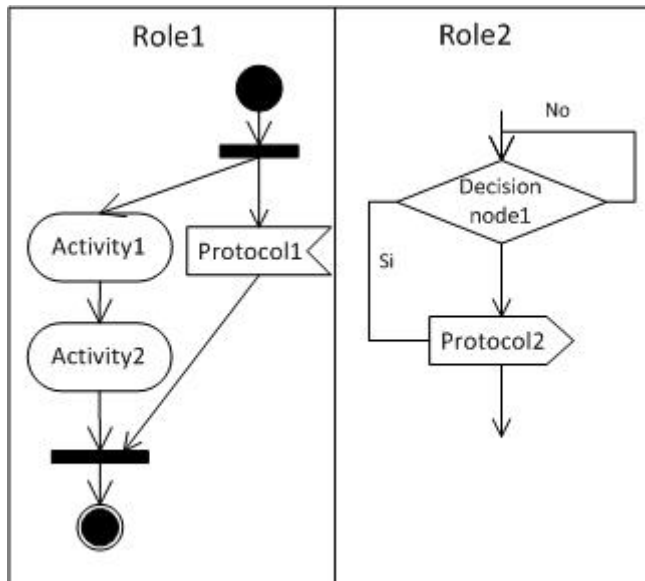
The UML Use Case Diagram is used as the graphical syntax to represent the Social Behavior Diagram (see Figure 6-12). The roles are represented as actors which are labeled with the stereotype «role». In addition, the social behavior relations are represented as relations labeled with the stereotypes «collaboration», «disposition» and «veracity». We propose naming the relations with the corresponding property (i.e. for the social behavior relation collaboration the relation is named as “communicative”, “non-communicative” or both, for the social behavior relation disposition the relation is named as

“benevolent”, “self-interested”, “malevolent” or the combinations, and finally for the social behavior relation veracity the relation is named as “truthful”, “untruthful” or both).

The UML Activity Diagram is used as the graphical syntax to represent Activity Diagram (see Figure 6-13). Regarding the parts we use in the activity diagram, we use the initial node, the end node, an activity, flow of control, branch elements (fork and join), decision node, activity partition (one for each role), send signal action (for initiator protocol), and accept event action (for responder protocol).



**Figure 6-12** Social Behavior Model Graphical Syntax: Social Behavior Diagram



**Figure 6-13** Social Behavior Model Graphical Syntax: Activity Diagram

## 6.6. Environment Model

The Environment Model represents the permissions of the roles identified in the Role Model with regard to the resources of the Domain Model. For each role identified in the Role Model, resources are established for those who can legitimately access them. Finally, the permissions (perceive or modify) are established.

Figure 6-14 shows the Environment Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Role Model, and Domain Model. In Figure 6-14 it is possible to observe that the *EnvironmentModel* metaclass is a container for the elements of this model. The *EnvironmentModel* metaclass is composed of resources, permissions and roles, represented respectively by the aggregation relations between the *EnvironmentModel* metaclass and the *Resource*, and *PermissionRelation* metaclasses and *Role* metaclass from Role Model. The permissions are represented as relations between the role and the resource it can access, and can be of two types: perceive or modify. This is represented as one association relation between the *PermissionRelation* metaclass and *Role* metaclass from the Role Model, and one association relation between the *PermissionRelation* and *Resource* metaclasses, in which the source of the relation is represented with the association labeled *from*, and the target of the relation is represented with the association labeled *to*. The *Resource* metaclass has a relation with the *Entity* metaclass from the Domain Model, to represent the evolution of an entity from the Domain Model into a resource in the Environment Model.

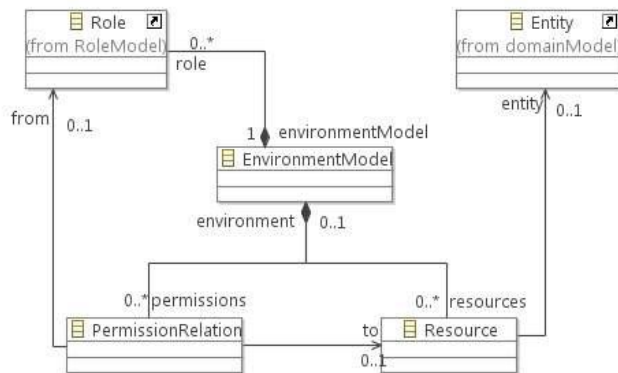
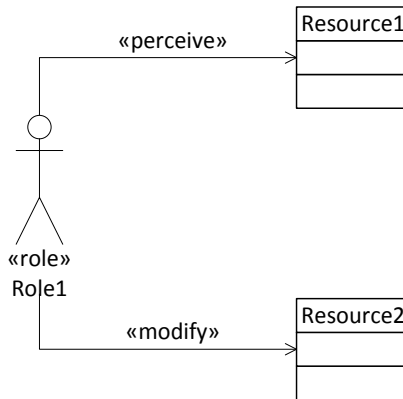


Figure 6-14 Environment Model

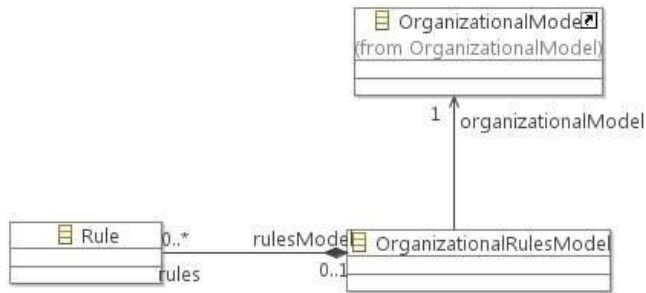
The UML Use Case Diagram is used as the graphical syntax to represent this information (see Figure 6-15). The roles are represented as actors which are labeled with the stereotype «role», the resources are represented as classes, and the permissions as relations between the role and the entity, which are labeled with the stereotypes «perceive», and «modify».



**Figure 6-15** Environment Model Graphical Syntax

## 6.7. Organizational Rules Model

The Organizational Rules Model represents the constraints of the organization's behavior. These rules can be viewed as general rules, responsibilities, restrictions, the desired behavior, and the sequence or order in such conduct. Three types of rules are distinguished: precedence, restriction and general. Figure 6-16 shows the Organizational Rules Model metaclasses and the relations with metaclasses of other models of the proposal, in this case with the Organizational Model. The *OrganizationalRulesModel* metaclass is a container for the elements of this model (see Figure 6-16). It is composed of a set of rules that form the model. This is represented as an aggregation relation between the *OrganizationalRulesModel* and *Rule* metaclasses. The relation from the *OrganizationalRulesModel* metaclass and the *OrganizationalModel* metaclass from the Organizational Model represents that a relation between them must exist.



**Figure 6-16** Organizational Rules Model

We propose a table schema for the graphical syntax of this model to represent a set of rules in which each rule is defined by a natural language description of the relationship, the type of rule, and the corresponding formula if necessary (see Table 6-1).

**Table 6-1** Organizational Rules Model Graphical Syntax

Rule	Type	Formula
Description rule 1	Precedence	$G1 < G2$
Description rule 2	Restriction	$G2 \rightarrow G4$
Description rule 2	General	

# Chapter 7

## Requirements for MAS Modeling

### Process

Software applications are complex products very difficult to develop and verify. For this reason, researchers put special attention on understanding and improving the quality of developed software. One of these research directions is based on the study and improvement of the software development process. A direct relationship between the process quality and the software quality it assumes. The research area related to these aspects refers to these processes using the term *processing software*.

A software process is a coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to design, develop, install and maintain a software product [27]. A software development process aims at an effective and efficient production of a software product that meets customer needs. The software process models facilitate the understanding of the project to be carried out.

On the other hand, [63] defines software process model as: “A simplified representation of a software process, represented from a specific perspective. By their simplified nature, a software process model is an abstraction of a real process”.

This chapter presents the process of development proposed for modeling the requirements of a MAS, using SPEM 2.0. This guide covers both phases of definition and specification of system requirements.

#### 7.1. Introduction to SPEM 2.0

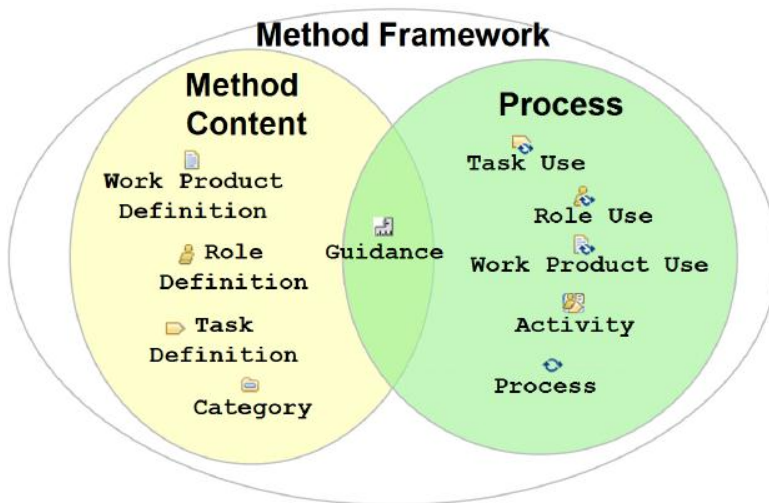
SPEM 2 (Software & Systems Process Engineering Metamodel Specification, v2.0) [56] is a metamodel used to define software and systems development processes and their components. The scope of SPEM is purposely limited to the minimal elements necessary to define any software and systems development process, without



adding specific features for particular development domains or disciplines. The goal is to accommodate a large range of development methods and processes of different styles, cultural backgrounds, levels of formalism, lifecycle models, and communities. However, the focus of SPEM is development projects.

SPEM 2.0 does not aim to be a generic process modeling language, nor does it even provide its own behavior modeling concepts. SPEM 2.0 rather defines the ability for the implementer to choose the generic behavior modeling approach that best fits their needs. It also provides specific structures to enhance such generic behavior models that are characteristic for describing development processes.

The central idea of SPEM 2 to represent processes is based on three basic elements: role, work product and task. Tasks represent the effort to do, roles represent who does the work and work products represent the inputs used in the work and outputs produced. In this way, it is specified: *“who (roles) does what (tasks) in order to get outputs (work products) from inputs (work products)”*.











**Figure 7-1** SPEM modeling primitives








When defining a process, SPEM 2 distinguishes two stages (see Figure 7-1):

- Define the essential elements that provide content to the process. These elements (roles, tasks, work product) compose the repository called Method Content;
- These elements are combined and reused to assemble processes and activities, defining the workflow between them.

Table 7-1 describes the most commonly used modeling primitives when defining a process.

**Table 7-1** Subset of elements to model processes in SPEM 2.0

Icon	Name	Description
	Process	Represents a complete process. Is a set of internally consistent process descriptions that can be reused to define major processes.
  	Activity Phase Iteration	Representing a set of tasks that run within the process, along with their roles and related products. If you only want to represent a group of tasks, you can use the "Activity" item or "Phase", or if it is a set of tasks that repeats a certain number of times, you can use the element "Iteration".
	Role Definition	Is a set of skills, capabilities and responsibilities of an individual or group.
	Task Definition	Describes a unit of work allocated and managed, identifying the work being performed by roles. Can be divided into several steps.
	Work Product Definition	It is the product used or produced by a "Task". There are two types of products: tangible nature artifact and package products for delivery. You can associate each other through aggregation relationships, composition and impact.
	Category	Classified items such as "Tasks", "Roles" and "Products" based on the criteria the process engineer wishes. There are various types of categories: "Set of Roles", "Discipline", and "Domain".

Icon	Name	Description
	Discipline	Collection of "Tasks" that are related to a major area of effort within an entire project.
	Role Set	Is used to group "Roles" that have something in common (e.g., they use similar techniques require similar skills)
	Domain	Allow to establish a hierarchy of domains, to classify "Work Products" with as many levels as desired.
	Role Use	Represents the role that performs a task or activity within a given process. Refers to a "Role Definition".
	Task Use	Represents an atomic task within a given process. Refers to a "Task Definition".
	Work Product Use	Represents a "Work Product" input or output related to an activity or task. Refers to a definition of a "Work Product".
	Process Package	Represents a package grouping all the elements of the process.

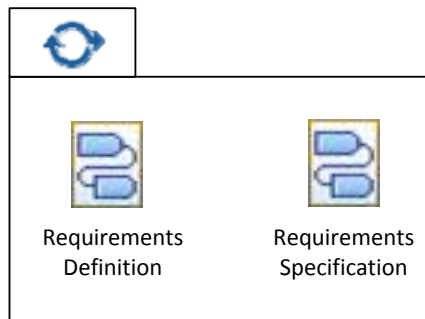
By using a framework such as that offered by SPEM 2 provides many advantages, since you can have software processes models in computer-processable format, which in turn provides capabilities for:

- Facilitate understanding and communication between people, because it provides a common framework where the concepts have a formal definition, thus promoting a uniform knowledge.
- Facilitate reuse, because the definition of a process can be integrated as parts or patterns of other process models.
- To support process improvement, because the formally defined activities facilitate their evaluation through measurement processes, which provide feedback to help improve these processes.
- To support processes management, because it provides a repository with the content of the process, which facilitates access by the various involve persons.

- To guide the process automation and support for automatic execution through the creation of workflows that can be implemented in tools.

## 7.2. Disciplines

As mentioned before, to define the *Method Content* the process engineer can classify items such as *Tasks*, *Roles* and *Products*. A discipline is defined as a collection of tasks related to an *area of interest* in the overall project. The partitioning of tasks in this way implies that the associated roles and resulting work products are categorized under the same theme. The grouping of tasks in disciplines helps to understand a project from the traditional waterfall project view.



**Figure 7-2** Disciplines of the Requirements Modeling Process for MAS

In our proposal there are two disciplines, as shown in Figure 7-2:

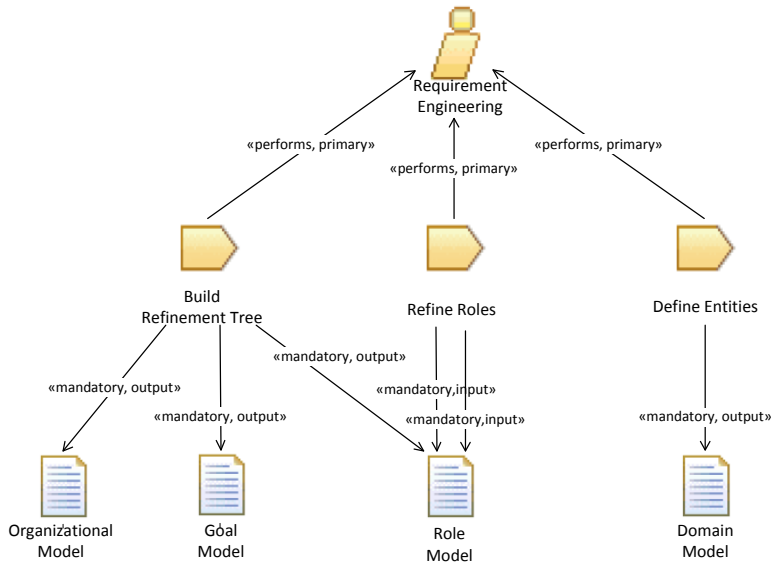
- Requirements Definition: this discipline includes all tasks necessary to build a requirement specification.
- Requirements Specification: the goal of this discipline is to refine the artifacts produced in the Requirements Definition discipline and finish building a final specification of requirements.

### 7.2.1 Requirements Definition Discipline

In the Requirements Definition discipline shown in Figure 7-3 the Requirement Engineering role is involved. This role will be responsible for carrying out the following activities: Build Refinement Tree, Refine Roles, and Define Entities.

In this discipline the work products are all models: Organizational Model, Role Model, Goal Model, and Domain Model.

As can be seen in the Figure 7-3, the outputs of the task Build Refinement Tree are the Organizational Model, Role Model and Goal Model, since the construction of the refinement tree is intended to represent the information of these three models. The task Refine Roles needs the Role Model as input, and the output is the Role Model updated. Finally, the output of the Define Entities task is the Domain Model.



**Figure 7-3** Requirement Definition Discipline

## 7.2.2 Requirements Specification Discipline

In the Requirements Specification discipline shown in Figure 7-4 the Requirement Engineering role is involved. This role will be responsible for carrying out the following activities: Create Social Behavior Model, Create Environment Model, and Define Organizational Rules.

In this discipline the work products are all models: Organizational Model, Role Model, Goal Model, Domain Model, Behavior Model, Environment Model, and Organizational Rules Model.

As can be seen in the Figure 7-4, the task Create Social Behavior Model needs the Organizational Model, Role Model, and Goal Model as input, and the output is the Behavior Model. The task

Create Environment Model needs the Role Model, and Domain Model as input, and the output is the Environment Model. Finally, the task Define Organizational Rules needs the Role Model as input, and the output is the Organizational Rules Model.

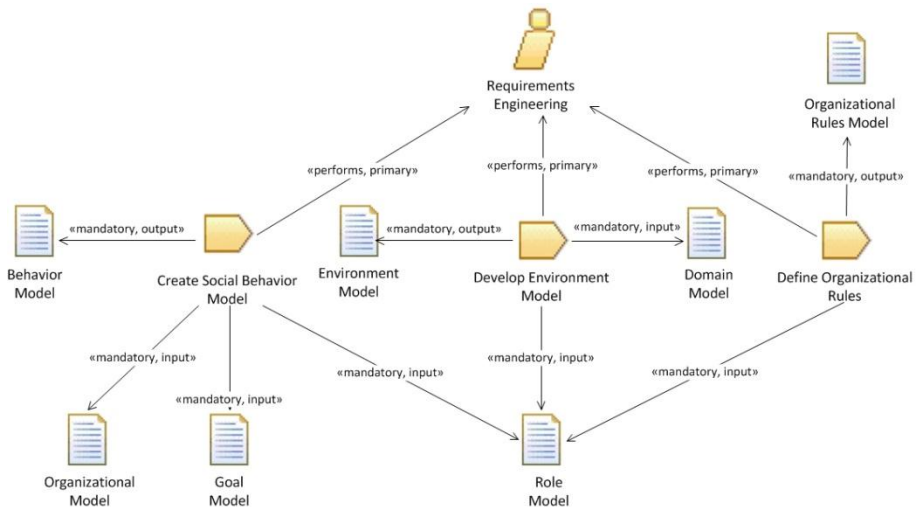
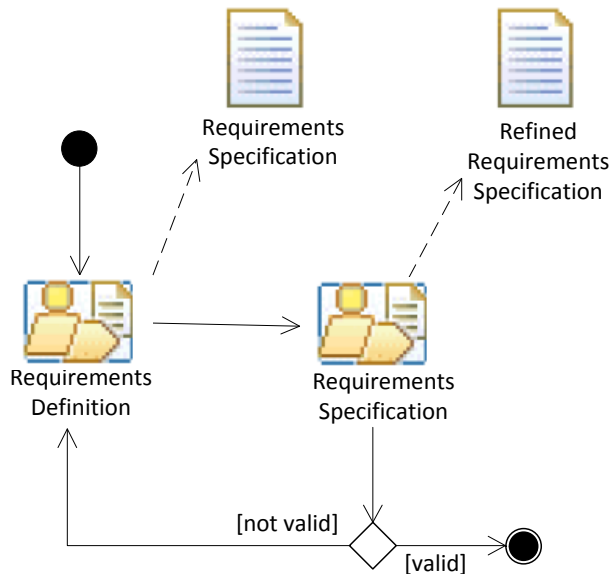


Figure 7-4 Requirement Specification Discipline

### 7.3. Process

As was mentioned earlier, the requirements modeling process proposed involves two phases: Requirements Definition and Requirements Specification. Figure 7-5 shows an overview of this process. Each activity of the process produces a document that is composed of the sum of all the models and documents of the working definition that is included in each activity.

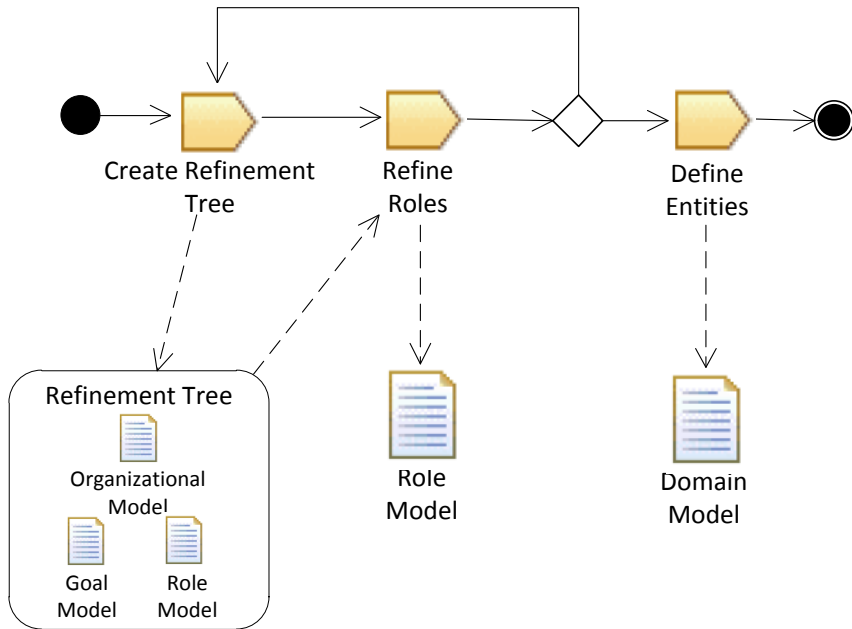
The Requirements Definition activity tasks are performed first, thus producing the requirements specification. The Requirements Specification activity tasks are then performed, using the requirements specification produced in the previous activity as input and resulting in the production of the refined requirements specification. At this point the Requirements Definition activity can again be performed in case some kind of inconsistency or incompleteness is encountered in the specification, or the process may end.



**Figure 7-5** Requirements modeling process overview

### 7.3.3 Requirements Definition

The Requirements Definition activity consists of three tasks whose aim is to identify the models of the phase, as is shown in Figure 7-6. The first task is to Create Refinement Tree, beginning with the definition of the Mission Statement, which is then broken into sub-organizations, roles and goals. This information is part of the Organizational Model, Role Model and Goal Model. The list of roles identified in the previous task will be used as input for the next task: Refine Roles. Here we discuss possible structural similarities in order to identify inheritance relationships. If deemed appropriate, it is possible to return to the previous task in order to update the Refinement Tree, or the next task can be performed. In the last task, Define Entities, the Domain Model is constructed from the identified entities, and association and inheritance relationships among them are defined.



**Figure 7-6** Requirements Definition activity decomposed into tasks and artifacts

### 7.3.4 Requirements Specification

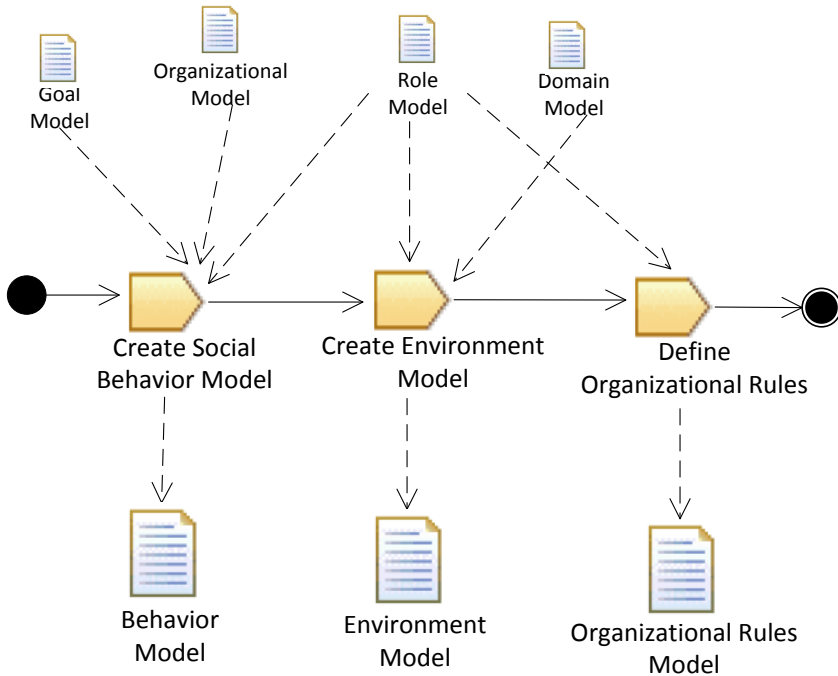
The Requirements Specification activity involves the creation of three models: Behavior Model, Environment Model and Organizational Rules Model, and therefore consists of three tasks for the creation of the models, as is shown in Figure 7-7.

The first task in this activity is Create Social Behavior Model. The Organizational Model, Role Model, Goal Model of the Requirements Definition activity are used as input. We analyze the goals to be attained by each role in each sub-organization in order to identify the social behavior relationships between them and the necessary Social Behavior Diagrams are created. Also, the necessary Activity diagrams are created as a result of analyzes the flow of tasks to be performed by a role to achieve each goal.

When this has been completed, the next task is performed: Create Environment Model. The Role Model and the Domain Model of the Requirements Definition phase are taken as input.



Then, the Define Organizational Rules task is performed, taking as input the Role Model of the Requirements Definition activity. The Organizational Rules Model is produced as a result of this. Finally, the artifacts generated during the process can relate to analysis and design artifacts from other methodologies by establishing a traceability framework. This will increase the overall quality of the system to be.



**Figure 7-7** Requirements Specification activity decomposed into tasks and artifacts

# Chapter 8

## Case Study

Agent technology is useful in many complex domains: ecommerce, health, stock market, manufacturing, games, etc. In particular, we are interested in the game development domain since it comprises a set of characteristics such as collaboration, negotiation, trust, reputation, etc., which specially can be related with a MAS. According to Google Trends and the ESA annual report [22], games development is one of the business markets that have undergone most growth in the last few years.

In addition, the agent-oriented paradigm is one of the most promising for modeling such business market due to the social behavior characteristics (negotiation, cooperation, etc.) of the agents and the complexity that MASs can support. For this reason, this chapter illustrates the feasibility of our approach by applying the requirements modeling process to the development of the strategic board Diplomacy Game [23].

### 8.1. Diplomacy Game

We have used the Diplomacy Game to verify the feasibility of our approach in areas such as negotiation, argumentation, trust and reputation [23] in the game development domain. Many interesting features make the Diplomacy Game compelling for applying the agent technology: the absence of random movements, all players move their units simultaneously, all units are equally strong so when one attacks another the winner of the battle is decided by considering solely the number of units helping one another, etc. Accordingly, from a player's point of view, the most important feature of the game is the negotiation process: deciding allies, selecting who to ask for help, arguing with other players to obtain information about their objectives or to discover what they know, and so on. We have used the rulebook of the Diplomacy Game [69] as a description of the system to be modeled with the process proposed in this work. The most relevant aspects of the game are provided as follows.

The Diplomacy Game is played by seven players and a Game Master. Each player represents one of the seven “Great Powers of Europe” in the years prior to World War I. These Great Powers consist of England, Germany, Russia, Turkey, Italy, France, and Austria. At the start of the game, the players randomly decide which Great Power each will represent. This is the only element of chance in the game. As soon as one Great Power controls 18 supply centers, it is considered to have gained control of Europe. The player representing that Great Power is the winner.

Diplomacy is a game of negotiations, alliances, promises kept, and promises broken. In order to survive, a player needs help from others. In order to win the game, a player must eventually stand alone. Knowing who to trust, when to trust them, what to promise, and when to promise it is the heart of the game.

At the beginning of each turn, the players meet together in small groups to discuss their plans and suggest strategies. Alliances between players are made openly or secretly, and orders are coordinated. Immediately following this period of “diplomacy,” each player secretly writes an order for each of his or her units on a slip of paper. When all the players have written their orders, the orders are simultaneously revealed, and are then all resolved. Some units are moved, some have to retreat, and some are removed. Resolving orders is the most challenging part of the rules and requires complete knowledge of the rules. Each turn represents six months of time. The first turn is called a Spring turn and the next a Fall turn. After each Fall turn, each Great Power must reconcile the number of units it controls with the number of supply centers it controls. At this time some units are removed and new ones are built. The purpose of the Game Master is to keep time for the negotiation sessions, collect and read orders, resolve issues, and make rulings when necessary. This role should be strictly neutral.

Each turn has a series of phases: (i) Spring four-phase turn: Diplomatic phase, Order Writing phase, Order Resolution phase, Retreat and Disbanding phase; (ii) Fall five-phase turn: Diplomatic phase, Order Writing phase, Order Resolution phase, Retreat and Disbanding phase, Gaining and Losing Units phase. After a Fall turn, if one Great Power controls 18 or more supply centers, the game ends and that player is declared the winner.

## 8.2. Diplomacy Game with Agents

Based on the tasks of the Requirements Definition and Requirements Specification activities proposed, and which were presented in the previous section, the development of the case study is presented below.

### 8.2.1 Requirements Definition

As explained in the previous chapter, the requirements modeling process starts with the Requirements Definition activity, which is composed of three tasks: Create Refinement Tree, Refine Role Model, and Create Entities (see Figure 7-6), developed below.

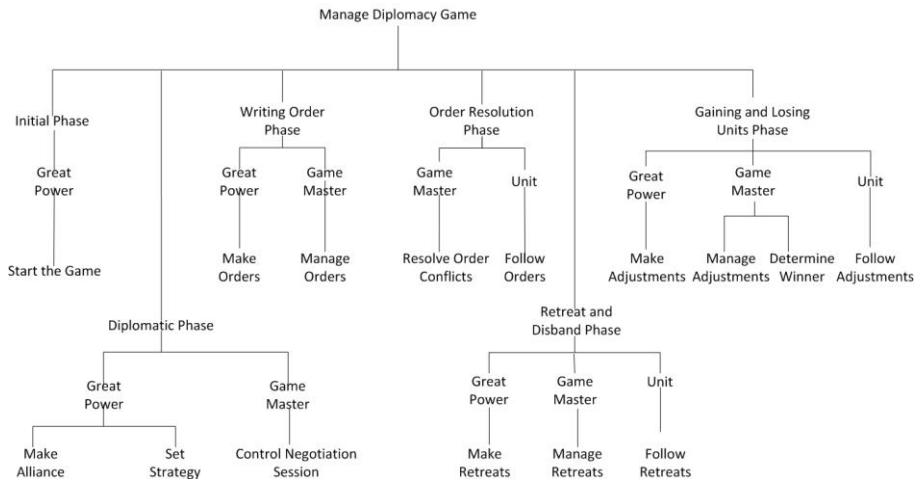
#### 8.2.1.1 Create Refinement Tree

This activity starts with the first task: Create Refinement Tree to represent the Organizational Model, partially the Role Model, and the Goal Model. First the Mission Statement of the system must be defined, which in this case is simple and is the *Management of the Diplomacy Game*. Then, the system Mission Statement has to be refined in sub-organizations. For the definition of the sub-organizations of the system we decided that the problem naturally leads to a conception of the whole system as a number of different MAS sub-organizations, one for each phase of the game, and one extra sub-organization representing the start of the game. The resulting sub-organizations are: *Initial phase*, *Diplomatic phase*, *Writing Order Phase*, *Order Resolution phase*, *Retreat and Disband phase* and *Gaining and Losing Units phase*. This concept of representing the sub-organizations of the system as phases was also used in the Conference Management System case study presented in [73].

The roles that are part of each sub-organization are then defined, resulting in three roles: *Great Power*, *Game Master* and *Unit* which, depending on which sub-organization they are, have different goals.

Finally the roles are refined with the goals they need to attain in order to fulfill each sub-organization's objective. In the *Initial Phase* sub-organization, the *Great Power* role has the goal of *Start the Game*. In the *Diplomatic Phase* sub-organization, the *Great Power* role has the goals of *Male Alliance*, and *Set Strategy*. In the same sub-organization

the *Game Master* role has the goal of *Control Negotiation Session*. In the *Writing Order Phase* sub-organization the *Great Power* role has the goal of *Make Orders*, and the *Game Master* role has the goal of *Manage Orders*. In the *Order Resolution Phase* sub-organization the *Game Master* role has the goal of *Resolve Order Conflicts*, and the *Unit* role has the goal of *Follow Orders*. In the *Retreat and Disband Phase* sub-organization the *Great Power* role has the goal of *Make Retreats*, the *Game Master* role has the goal of *Manage Retreats*, and the *Unit* role has the goal of *Follow Retreats*. Finally, in the *Gaining and Losing Units Phase* sub-organization the *Great Power* role has the goal of *Make Adjustments*. The *Game Master* role has the goals of *Manage Adjustments* and *Determine Winner*. The *Unit* role has the goal of *Follow Adjustments*. Figure 8-1 shows the complete resulting Refinement Tree.



**Figure 8-1** Diplomacy Game Refinement Tree

### 8.2.1.2 Refine Role Model

The second task, **Refine Role Model**, is performed to complete the Role Model based on the information defined in the Refinement Tree. Possible inheritance relationships between roles can be specified in this task. The current case study presents no inheritance relations, just three roles without hierarchy (see Figure 8-2).

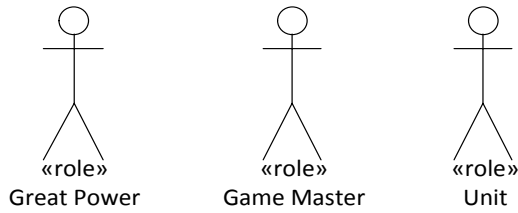


Figure 8-2 Diplomatic Game Inheritance Diagram

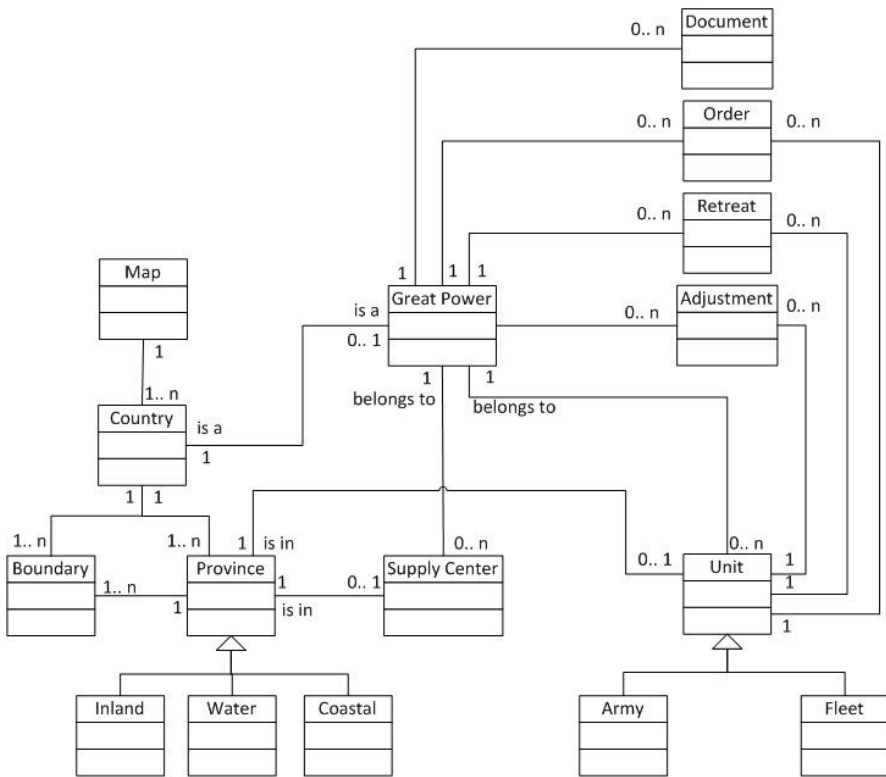


Figure 8-3 Diplomatic game Domain Model

8.2.1.3 Create Entities

The third task, **Create Entities**, is performed to define the Domain Model. Figure 8-3 shows the Domain Model generated. Briefly, the domain consists of a *Map* that is composed of many *Countries*, which in turn have *Boundaries* and *Provinces*. A *Province* can be an *Inland*, *Coastal* or *Water* province. A *Supply Center* is in a *Province*, but a *Province* may or may not have a *Supply Center*. Furthermore, a *Unit* is

in a *Province*, but a *Province* may or may not have a *Unit*. A *Unit* can be an *Army* or a *Fleet*. Both a *Province* and a *Unit* belong to a *Great Power*, which in turn is a *Country*, but not all *Countries* are *Great Powers*. A *Great Power* has many *Documents*, *Orders*, *Retreats* and *Adjustments*, and they all belong to only one *Great Power*. *Orders*, *Retreats* and *Adjustments* are all for one *Unit* and a *Unit* follows many *Orders*, *Retreats* and *Adjustments*.

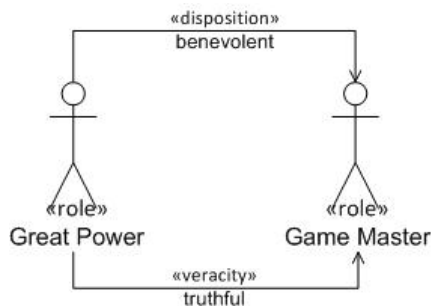
As a result of performing the Requirements Definition activity we obtain the Refinements Tree shown in Figure 8-1, which represents the Organizational Model, partially the Role Model and Goal Model. Also, we obtained an inheritance diagram, shown in Figure 8-2, to complement the information of the roles in the Refinement Tree. Finally, diagram relating the entities identified in the domain to represent the Domain Model is shown in Figure 8-3.

## 8.2.2 Requirements Specification

The second activity, in the requirements modeling process, is to perform the Requirements Specification, which is composed of three tasks: Create Social Behavior Model, Create Environment Model, and Define Organizational Rules (see), developed below.

### 8.2.2.1 Create Social Behavior Model

This activity starts with the first task, **Create Social Behavior Model**, in order to specify the Behavior Model using the information from the Organizational Model, Role Model, and Goal Model generated in the Requirements Definition activity as input. This task is composed in turn of two parts, first a social behavior diagram must be created for each sub-organization, and then an activity diagram must be created for each goal.

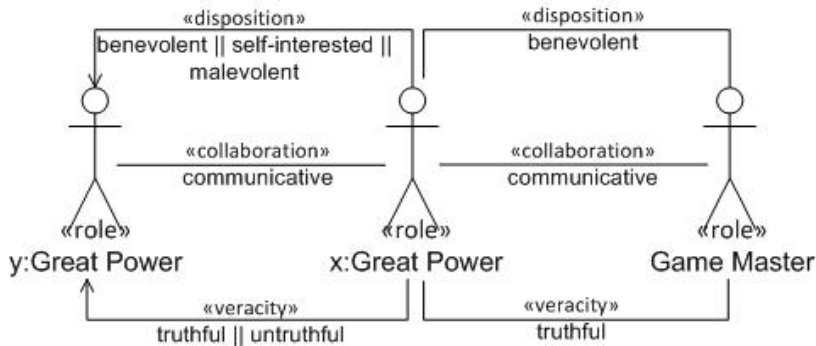


**Figure 8-4** Social behavior diagram of the Initial Phase sub-organization

The goals of each role in each sub-organization are reviewed in order to identify whether the role needs social behavior relationships in any sub-organization. The social behavior relationships can be of three types of social behavior with their respective properties: collaboration (communicative, non-communicative), disposition (benevolent, self-interested, malevolent), and veracity (truthful, untruthful). Following a social behavior diagram for each sub-organization identified in the case study is presented.

Upon analyzing the goal of the role of the *Initial Phase* sub-organization, we identified that the *Great Power* role needs to have a disposition relation to attain its *Start the Game* goal, and more specifically, the role needs to be *benevolent* with the *Game Master* role.

Also the *Great Power* role needs to have a disposition relation to attain its *Start the Game* goal, and more specifically, the role needs to be *truthful* with the *Game Master* role. Figure 8-4 shows the social relation diagram for the organization analyzed.



**Figure 8-5** Social behavior diagram of the Diplomatic Phase sub-organization

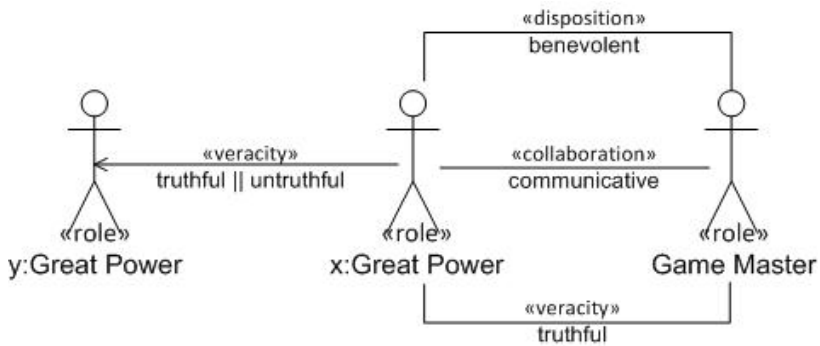
Upon analyzing the goals of the roles of the *Diplomatic Phase* sub-organization, we identified that the *Great Power* role needs to have the collaboration relation to attain all of its goals in the sub-organization analyzed, and more specifically, the role needs to be *communicative* with other instances of the *Great Power* role and with the *Game Master* role. The same applies in the case of the *Game Master* role fulfilling its *Control Negotiation Session* goal; the collaborative relationship will be with the *Great Power* role. The collaborative



relationship between *Great Power* and *Game Master* will therefore be on both sides, represented with a non-directional arrow.

Moreover, if the *Great Power* role is to fulfill all of its goals in the sub-organization analyzed, it needs to have a disposition relation; more specifically, it needs to be *benevolent*, *self-interested* or *malevolent* with regard to another instance of the *Great Power* role, depending on the agent's intentions. In this sub-organization, negotiation, persuasion and trust are keys to the *Great Power* role. Also, the *Great Power* role in the sub-organization analyzed is in all cases *benevolent* with regard to the *Game Master* role, and vice versa.

Finally, if the *Great Power* role is to fulfill all of its goals in the sub-organization analyzed, it is necessary for the *verity* relation between the *Great Power* role and other instance of the same role to be *truthful* or *untruthful*, again depending on the intentions of the agent playing the role. We also believe that it is necessary for the *verity* relation between the *Great Power* role and the *Game Master* role to be *truthful* in both directions. Figure 8-5 shows the social relation diagram for the organization analyzed.



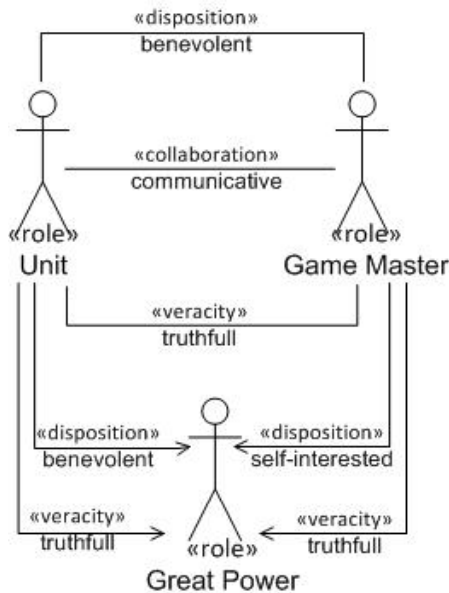
**Figure 8-6** Social behavior diagram of the Writing Order Phase sub-organization

Upon analyzing the goal of the role of the *Writing Order Phase* sub-organization, we identified that the *Great Power* role needs to have a collaboration relation to attain its *Make Order* goal, and more specifically, the role needs to be *communicative* with the *Game Master* role. The same applies in the case of the *Game Master* role fulfilling its *Manage Order* goal; the collaborative relationship will be with the *Great Power* role. The collaborative relationship between *Great Power*

and *Game Master* will therefore be on both sides, represented with a non-directional arrow.

Moreover, if the *Great Power* role is to fulfill its goal in the sub-organization analyzed it needs to have a disposition relation, more specifically, it needs to be *benevolent* with regard to the *Game Master* role, and vice versa.

Finally, if the *Great Power* role is to fulfill its goal in the sub-organization analyzed, it is necessary for the veracity relation between the *Great Power* role and other instance of the same role to be *truthful* or *untruthful*, again depending on the intentions of the agent playing the role. We also believe that it is necessary for the veracity relation between the *Great Power* role and the *Game Master* role to be *truthful* in both directions. Figure 8-6 shows the social relation diagram for the organization analyzed.



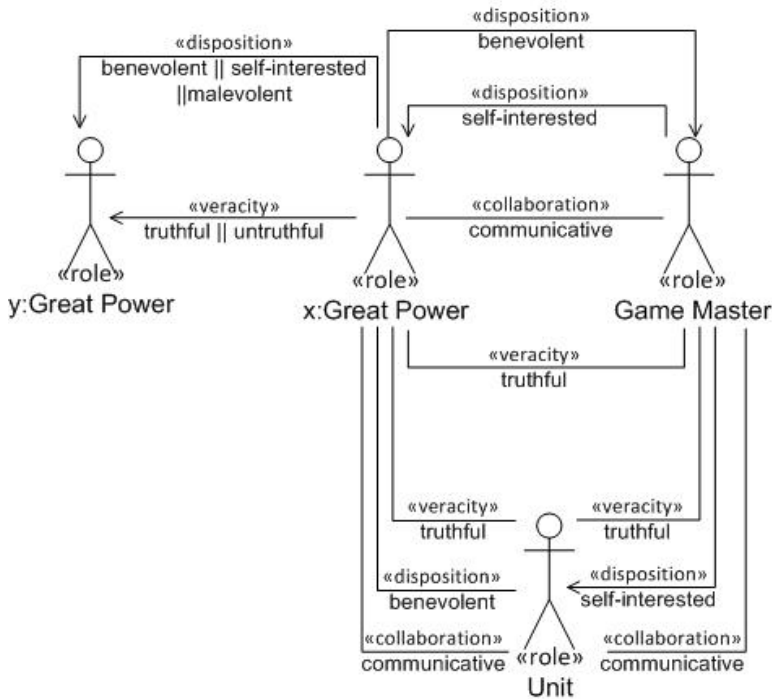
**Figure 8-7** Social behavior diagram of the Order Resolution Phase sub-organization

Upon analyzing the goal of the role of the *Order Resolution Phase* sub-organization, we identified that the *Game Master* role needs to have a collaboration relation to attain its *Resolve Order Conflicts* goal, and more specifically, the role needs to be *communicative* with the *Unit* role. The same applies in the case of the *Unit* role fulfilling its

*Follow Orders* goal; the collaborative relationship will be with the *Game Master* role. The collaborative relationship between *Game Master* and *Unit* will therefore be on both sides, represented with a non-directional arrow.

Moreover, if the *Game Master* role is to fulfill its goal in the sub-organization analyzed it needs to have a disposition relation, more specifically; it needs to be *benevolent* with regard to the *Unit* role, and vice versa. Also, the *Game Master* role needs to be *self-interested* with regards to the *Great Power* role. And, the *Unit* role needs to be *benevolent* with regards to the *Great Power* role.

Finally, if the *Game Master* role is to fulfill its goal in the sub-organization analyzed, it is necessary for the veracity relation between the *Game Master* role and the *Great Power* role to be truthful. Also, the *Unit* role needs to be *benevolent* with regards to the *Great Power* role. We also believe that it is necessary for the veracity relation between the *Game Master* role and the *Unit* role to be truthful in both directions. Figure 8-7 shows the social relation diagram for the organization analyzed.



**Figure 8-8** Social behavior diagram of the *Retreat and Disband Phase* and *Gaining and Loosing Units Phase* sub-organizations

Upon analyzing the goals of the roles of the *Retreat and Disband Phase* sub-organization, we identified that the *Great Power* role needs to have the collaboration relation to attain its *Make Retreat* goal, and more specifically, the role needs to be *communicative* with the *Game Master* role and with the *Unit* role, and vice versa. The same applies in the collaboration relation between the *Game Master* role and the *Unit* role, to fulfill its *Manage Retreat* goal and *Follow Retreat* goal, respectively.

Moreover, if the *Great Power* role is to fulfill its goal in the sub-organization analyzed, it needs to have a disposition relation, more specifically, it needs to be *benevolent*, *self-interested* or *malevolent* with regard to another instance of the *Great Power* role, depending on the agent's intentions. Also, the *Great Power* role in the sub-organization analyzed is in all cases *benevolent* with regard to the *Game Master* role, and is also in all cases *benevolent* with regards to the *Unit* role, and

vice versa. On the other hand, the *Game Master* role is self-interested with regards to the *Great Power* role and the *Unit* role.

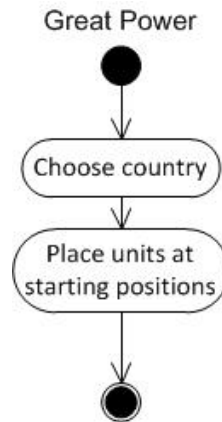
Finally, if the *Great Power* role is to fulfill all of its goals in the sub-organization analyzed, it is necessary for the veracity relation between the *Great Power* role and other instance of the same role to be *truthful* or *untruthful*, again depending on the intentions of the agent playing the role. We also believe that it is necessary for the veracity relation between the *Great Power* role and the *Game Master* role, and between the *Great Power* role and the *Unit* role to be *truthful* in both cases in both directions. The same applies in the veracity relation between the *Game Master* role and the *Unit* role. Figure 8-8 shows the social relation diagram for the organization analyzed.

Upon analyzing the goals of the roles of the *Gaining and Losing Units* Phase sub-organization, we identified that the collaboration, disposition, and veracity relation between the *Great Power*, *Game Master*, and *Unit* roles are the same as the social behavior relations shown in Figure 8-8.

The second part of the tasks **Create Social Behavior Model** is to analyze the goals of each role in each sub-organization in order to identify the sequence of steps that represent the flow of activities needed to achieve these goals. Following, the activity diagrams identified in the case study will be illustrated, and presented according to the order of sub-organizations.

Upon analyzing the goal of the *Initial Phase* sub-organization, we identified one activity diagram that specifies the activities and protocols performed by the *Great Power* role to attain the *Start the Game* goal.

As is shown in Figure 8-9, the flow of actions performed by the *Great Power* active role to attain the *Start the Game* goal begins with performing the *Choose county* activity, and second and last the *Place units at starting positions* activity.



**Figure 8-9** Activity Diagram of the *Start the Game* goal

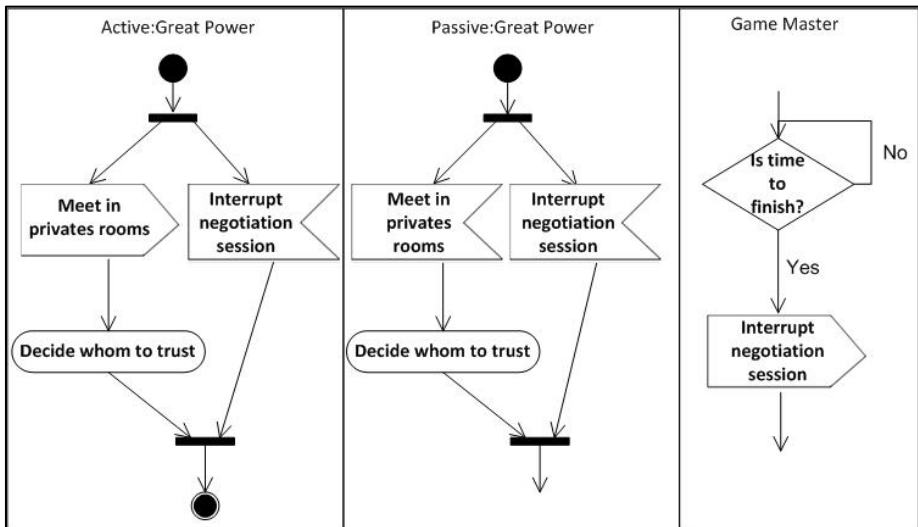
Upon analyzing the goal of the *Diplomatic Phase* sub-organization, we identified two activity diagrams that specify: i) the activities and protocols performed by the *Great Power* role to attain the *Make Alliance* goal, and the activities and protocols performed by the *Game Master* role to attain the *Control Negotiation Sessions* goal; ii) the activities and protocols performed by the *Great Power* role to attain the *Set Strategy* goal, activities and protocols performed by the *Game Master* role to attain the *Control Negotiation Sessions* goal.

As the *Make Alliance* goal and the *Set Strategy* goal are related to the *Control Negotiation Session* goal, we decide to specify their activity diagrams in the following way: i) one diagram with tree swim lines, two for the interaction between the two instances of the *Great Power* role (active and passive) to attain the *Make Alliance* goal, and the third swim line for the interaction between the *Game Master* role and the instances of the *Great Power* role to attain the *Control Negotiation Sessions* goal; ii) one diagram with two swim lines, one for the *Set Strategy* activities and protocols, and one for the interaction between the *Game Master* role and the instance of the *Great Power* role to attain the *Control Negotiation Sessions* goal.

As is shown in Figure 8-10, the flow of actions performed by the *Great Power* active role to attain the *Make Alliance* goal begins with a fork that gives the control to one initiator protocol: *Meet in private groups*, and to one reactive protocol: *Interrupt negotiation session* (reaction to the *Game Master* active protocol). The first protocol is

initialized by the *Great Power* active role and result in the reactive protocol *Meet in private groups* (Active:Great Power) of the *Great Power* passive role, while the other is a reaction of the *Great Power* role to the *Interrupt negotiation session* protocol initialized by the *Game Master* role if the negotiation time has ended. If this protocol is performed, the *Great Power* active role must terminate the flow of action.

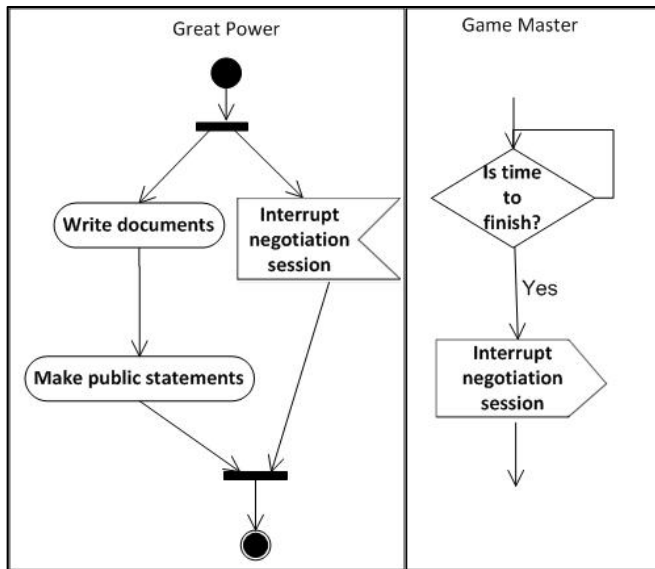
After the *Meet in private groups* protocol has been performed, the *Great Power* active role must perform the *Decide who to trust* activity in order to attain the *Make Alliance* goal. The *Great Power* passive role has the same flow of actions as the *Great Power* active role, with the difference that its *Meet in private groups* protocol is a reaction to the *Meet in private groups* protocol initialized by the *Great Power* active role, and since this is a passive instance of the *Great Power* role, it does not end the flow of actions.



**Figure 8-10** Activity Diagram for the goals *Make Alliance* and *Control Negotiation Session*

As is shown in Figure 8-11, the flow of actions performed by the *Great Power* role to attain the *Set Strategy* goal begins with a fork that gives the control to one activity: *Write Documents*, and to one reactive protocol: *Interrupt negotiation session* (reaction to the *Game Master* active protocol). After the *Write documents* activity has been performed, the *Great Power* role must perform the *Make public statements* activity in order to attain the *Set Strategy* goal. The reactive

protocol is a reaction of the *Great Power* role to the *Interrupt negotiation session* protocol initialized by the *Game Master* role if the negotiation time has ended. If this protocol is performed, the *Great Power* active role must terminate the flow of action.

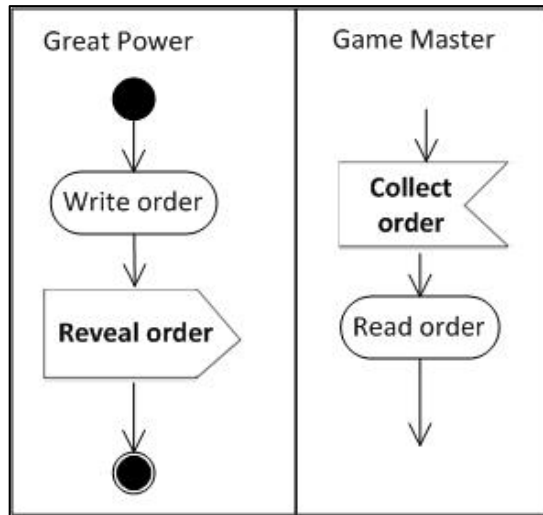


**Figure 8-11** Activity Diagram for the goals *Set Strategy* and *Control Negotiation Session*

Upon analyzing the goal of the *Writing Order Phase* sub-organization, we identified one activity diagrams that specify the activities and protocols performed by the *Great Power* role to attain the *Make Order* goal, and the activities and protocols performed by the *Game Master* role to attain the *Manage Order* goal. As the *Make Order* goal and the *Manage Order* goal are related, we decide to specify their activity diagrams in one diagram with two swim lanes for the interaction between the *Great Power* role and *Game Master* role to attain the *Make Order*, and *Manage Order* goals, respectively.

As is shown in Figure 8-12, the flow of actions performed by the *Great Power* role to attain the *Make Order* goal begins with the *Write documents* activity, and following with the *Reveal order* active protocol. The active protocol generates a reaction protocol of the *Game Master* role: *Collect order*. After the *Collect order* protocol has been performed, the *Game Master* role must perform the *Read order* activity in order to attain the *Manage Order* goal.

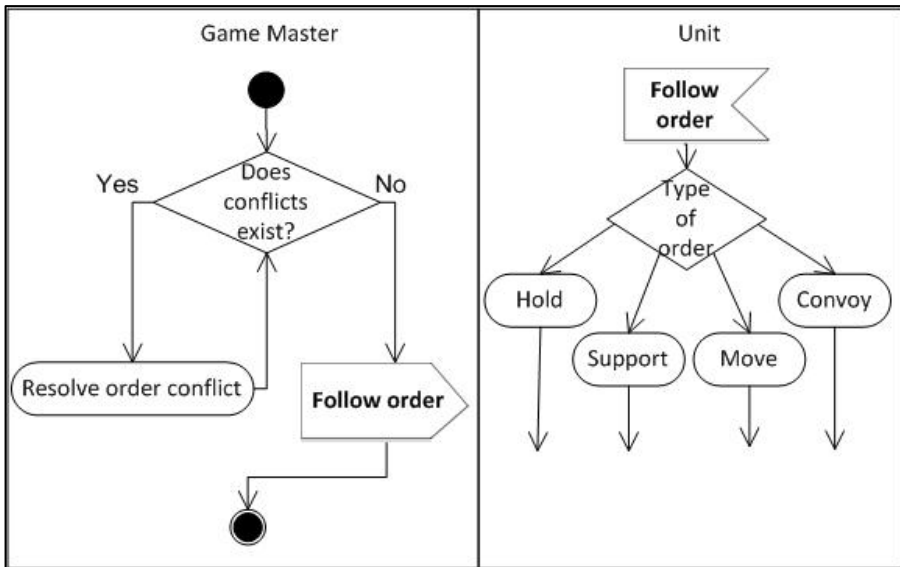




**Figure 8-12** Activity Diagram for the goals *Make Order* and *Manage Order*

Upon analyzing the goal of the *Order Resolution Phase* sub-organization, we identified one activity diagrams that specify the activities and protocols performed by the *Game Master* role to attain the *Resolve Order Conflicts* goal, and the activities and protocols performed by the *Unit* role to attain the *Follow Order* goal. As the *Resolve Order Conflicts* goal and the *Follow Order* goal are related, we decide to specify their activity diagrams in one diagram with two swim lines for the interaction between the *Game Master* role and *Unit* role to attain the *Resolve Order Conflicts*, and *Follow Order* goals, respectively.

As is shown in Figure 8-13, the flow of actions performed by the *Game Master* role to attain the *Resolve Order Conflicts* goal begins with the *Does conflicts exist?* decision node. If an order conflict does exist, the *Game Master* role must perform the *Resolver Order Conflict* activity; else, the *Game Master* role must perform the *Follow Order* active protocol in order to attain the *Resolve Order Conflicts* goal. The active protocol generates a reaction protocol of the *Unit* role: *Follow Order*, which in turn follows with the *Type of Order* decision node. Depending on the type of order, the *Unit* role must perform one of the following activities: *Hold*, *Support*, *Move*, or *Convoy* in order to attain the *Follow Orders* goal.

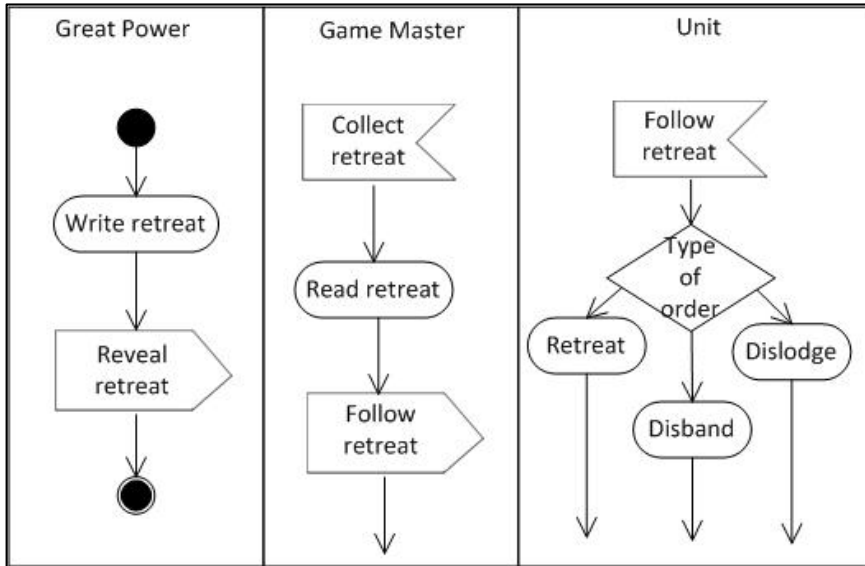


**Figure 8-13** Activity Diagram for the goals *Resolve Order Conflicts* and *Follow Order*

Upon analyzing the goal of the *Retreat and Disband Phase* sub-organization, we identified one activity diagram that specify the activities and protocols performed by the *Great Power* role to attain the *Make Retreats* goal, the activities and protocols performed by the *Game Master* role to attain the *Manage Retreats* goal, and the activities and protocols performed by the *Unit* role to attain the *Follow Retreats* goal. As these three goals are related, we decide to specify their activity diagrams in one diagram with three swim lines for the interaction between the *Great Power* role and *Game Master* role to attain the *Make Retreats*, and *Manage Retreats* goals, respectively, and for the interaction between the *Game Master* role and *Unit* role to attain the *Manage Retreats*, and *Follow Retreats* goals, respectively.

As is shown in Figure 8-14, the flow of actions performed by the *Great Power* role to attain the *Make Retreats* goal begins with the *Write retreat* activity, and following with the *Reveal retreat* active protocol. The active protocol generates a reaction protocol of the *Game Master* role: *Collect retreat*. After the *Collect retreat* protocol has been performed, the *Game Master* role must perform the *Read retreat* activity. Then, the *Game Master* role must perform the *Follow Order* active protocol in order to attain the *Manage Retreats* goal. The active

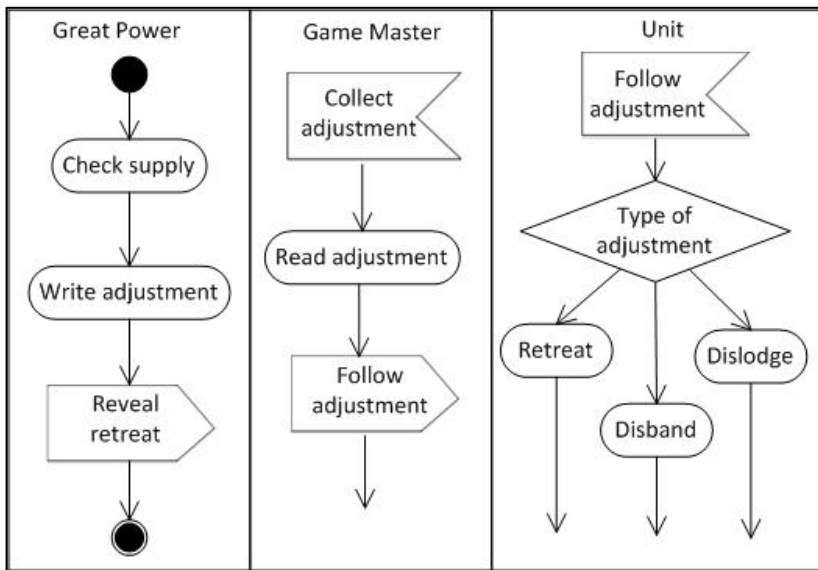
protocol generates a reaction protocol of the *Unit* role: *Follow retreat*, which in turn follows with the *Type of retreat* decision node. Depending on the type of retreat, the *Unit* role must perform one of the following activities: *Retreat*, *Disband*, or *Dislodge* in order to attain the *Follow Retreats* goal.



**Figure 8-14** Activity Diagram for the goals *Make Retreats*, *Manage Retreats* and *Follow Retreats*

Upon analyzing the goal of the *Gaining and Losing Units Phase* sub-organization, we identified two activity diagrams: i) one activity diagram that specify the activities and protocols performed by the *Great Power* role to attain the *Make Adjustments* goal, the activities and protocols performed by the *Game Master* role to attain the *Manage Adjustments* goal, and the activities and protocols performed by the *Unit* role to attain the *Follow Adjustments* goal. As these three goals are related, we decide to specify their activity diagrams in one diagram with three swim lines for the interaction between the *Great Power* role and *Game Master* role to attain the *Make Adjustments*, and *Manage Adjustments* goals, respectively, and for the interaction between the *Game Master* role and *Unit* role to attain the *Manage Adjustments*, and *Follow Adjustments* goals, respectively; ii) one activity diagram that specify the activities and protocols performed by the *Game Master* role to attain the *Determine Winner* goal.

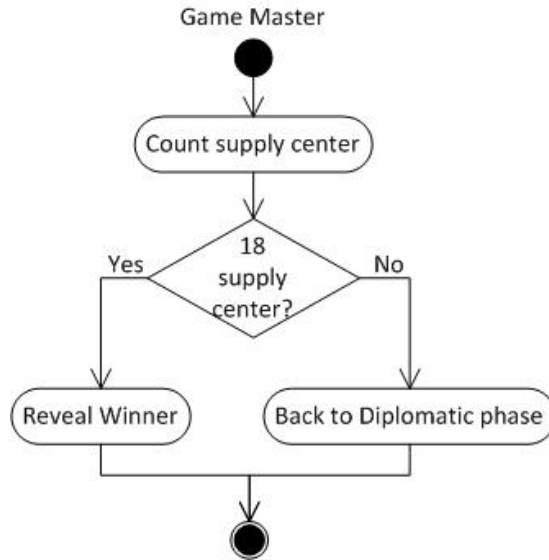
As is shown in Figure 8-15, the flow of actions performed by the *Great Power* role to attain the *Make Adjustments* goal begins with *Check supply* activity, and based on the number of supply left the *Great Power* role must perform the *Write adjustment* activity, and following with the *Reveal adjustment* active protocol. The active protocol generates a reaction protocol of the *Game Master* role: *Collect adjustment*. After the *Collect adjustment* protocol has been performed, the *Game Master* role must perform the *Read adjustment* activity. Then, the *Game Master* role must perform the *Follow adjustment* active protocol in order to attain the *Manage Adjustments* goal. The active protocol generates a reaction protocol of the *Unit* role: *Follow adjustment*, which in turn follows with the *Type of adjustment* decision node. Depending on the type of adjustment, the *Unit* role must perform one of the following activities: *Retreat*, *Disband*, or *Dislodge* in order to attain the *Follow Adjustments* goal.



**Figure 8-15** Activity Diagram for the goals *Make Adjustments*, *Manage Adjustments* and *Follow Adjustments*

As is shown in Figure 8-16, the flow of actions performed by the *Game Master* role to attain the *Determine Winner* goal begins with the *18 supply center?* decision node. This decision node means that the *Game Master* role must count the supply centers left for each *Great Power* role, and if *Great Power* role does have 18 supply centers, the

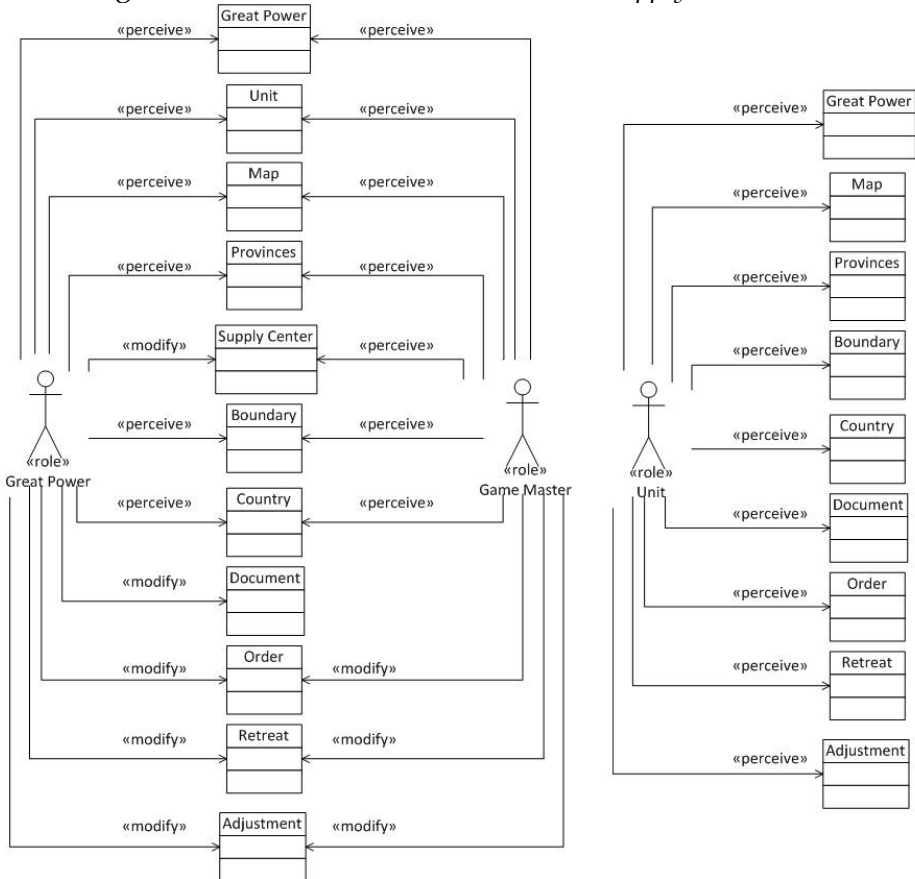
*Game Master* role must perform the *Reveal winner* activity; else, the *Game Master* role must perform the *Back to Diplomatic Phase* active protocol in order to attain the *Determine Winner* goal. If a winner is determine then the game is finish, else, the goals of the *Diplomatic Phase* sub-organization must be perform.



**Figure 8-16** Activity Diagram for the goals *Determine Winner*

The second task that must be performed in the Requirements Specification activity is **Create Environment Model** using the information from the Role Model and Domain Model generated in the Requirements Definition activity as input. Figure 8-17 shows the permissions of the *Great Power*, *Game Master* and *Unit* roles with regard to the Domain Model resources that each role needs to *perceive* or *modify* in order to attain its goals. The *Great Power* role *perceives* the following entities in the system: other instances of *Great Power*, *Units*, *Map*, *Provinces*, *Boundary* and *Country*; and can *modify*: *Supply Center*, *Document*, *Order*, *Retreat* and *Adjustment*. The *Game Master* role *perceives* the following entities in the system: *Great Power*, *Units*, *Map*, *Provinces*, *Supply Center*, *Boundary* and *Country*; and can *modify*: *Order*, *Retreat* and *Adjustment*; but cannot *perceive* or *modify* the *Document* entity. Finally the *Unit* role *perceives* the following entities in the system: *Great Power*, *Map*, *Provinces*, *Boundary*, *Country*, *Document*,

*Order, Retreat and Adjustment*; but cannot *perceive* or *modify* the following entities: other instances of *Unit* and *Supply Center*.



**Figure 8-17** Diplomacy game *Environment Model*

The third task that must be performed in the Requirements Specification activity is to **Define Organizational Rules**, using the information from the Domain Model generated in the Requirements Definition activity and the Behavior Model of the current activity as input. In the current domain, the important rules to identify are the general rules of the game, the number of players, the rules concerning the movement of the units depending on the type of unit and on the type of provinces the move take place in, etc. Table 1 shows an extract from the Organizational Rules Model.

**Table 8-1** *Organizational Rules Model*

<b>Rule</b>	<b>Type</b>	<b>Formula</b>
The game is divided into a two year tour: Spring four-phase turn and Fall five-phase turn	General	
Spring four-phase turn has phases: Diplomatic, Order Writing, Order Resolution and Retreat and Disbanding	General	
Fall five-phase turn has phases: Diplomatic, Order Writing, Order Resolution and Retreat and Disbanding, Gaining an Losing	General	
Only seven players may perform the role of "Great Power"	General	
When 18 supply centers belongs to a "Great Power" the game ends and the winner is that "Great Power"	General	
At the start of the game each "Great Power", except Russia, controls 3 supply centers	General	
At the start of the game, the "Great Power" Russia controls 4 supply centers	General	
Maximum time in the first diplomatic phase is 30 minutes	General	
Maximum time in the next diplomatic phase is 15 minutes	General	
All units have the same strength	General	
Only fleets can be ordered to convoy	General	
An army can be ordered to move into an adjacent inland or coastal province, not to a water provinces	General	
A fleet can be ordered to move to an adjacent water province or coastal province, not to an inland province	General	
Any location in the map that isn't named can't be occupied	General	
Islands can't be occupied, with the exception of England	General	
Units of equal strength trying to occupy the same province cause all units to remain in their original provinces, this is call a "standoff"	General	
A standoff doesn't dislodge a unit already in the province where the standoff took place	General	
One unit not moving can stop a unit or series of units from moving	General	
Units can't trade places without the use of convoy	General	
Three or more units can rotate provinces during a turn provided none directly trade places	General	
The province that a unit is providing support to must be one that the supporting unit could have legally	General	

moved to during the turn		
A unit not ordered to move can be supported by a support order that only mentions its provinces	General	
A unit ordered to move can only be supported by a support order that matches the move the unit is trying to make	General	
A dislodge unit can still cause a standoff in a province different from the one that dislodge it	General	
A dislodge unit, even with support, has no effect on the province that dislodge it	General	
Support is cut if the unit giving support is attacked from any province except the one where support is being given	General	
Support is cut if the unit giving support is dislodged	General	
A unit being dislodged by one province can still cut support in another province	General	
A Fleet in a water province can convoy an Army from any coastal province adjacent to that water province to any other coastal province adjacent to that water province	General	
For a Fleet to Convoy an Army, the Army must be ordered to move to the intender province and the Fleet must be ordered to convoy it	General	
A Fleet can't convoy more than one Army during the same turn	General	
Only Army can be convoy	General	
Support can't be transported from one Army via a convoy to another unit	General	
If Fleet occupy adjacent water provinces, an Army can be convoyed through all these water provinces on one turn	General	
Dislodgment of a Fleet in a convoy causes the convoy to fail	General	
A convoy that causes the convoyed Army to standoff at its destination results in that Army remaining in its original provinces	General	
A country can't dislodge or support the dislodgment of one of its own units, even if that dislodgment is unexpected	General	
A country can create a standoff by ordering two equally-supported attacks on the same province	General	
An attack by a country on one of its own units doesn't cut support	General	
Two units can exchange places if either or both are convoyed	General	



An army convoyed using alternate convoy orders reaches its destination as long as at least one convoy route remains open	General	
A convoyed Army doesn't cut the support of a unit supporting an attack against one of the Fleets necessary for the Army to convoy	General	
An Army with at least one successful convoy route will cut the support given by a unit in the destination province that is trying to support an attack on a Fleet in an alternate route of the convoy	General	
A dislodge unit can retreat to an adjacent province that it could ordinarily move to if unopposed by other units	General	
A unit can't retreat to: a province that is occupied; the province from which the attacker came; a province that was left vacant by a standoff during the same turn	General	
If there is no available province to retreat to, the dislodged unit is immediately disbanded and removed from the map	General	
Retreats can't be convoy or supported	General	
If one or more units are ordered to retreat to the same provinces, they all must be disbanded	General	
If a player fails to order a retreat when necessary, the unit is disbanded	General	
A country controls a supply center when one of its units occupies that supply center province after a turn has been played and completed	General	
If a country has more supply centers than units, it must disband the excess number of units	General	
If a country has more supply centers than units, it can place new units in each unoccupied supply center of its home country that it still controls	General	
A country can't build a supply center outside its home country	General	

Finally, as a result of performing the Requirements Specification activity we obtain the Behavior Model which is composed with: all the Social Behavior diagrams (e.g. Figure 8-5), and all the Activity diagrams (e.g. Figure 8-11). We also obtain the Environment Model (see Figure 8-17) as well as a table representing the Organizational Rules Model (see Table 8-1).

### 8.3. Discussion

With the definition of the Diplomacy Game Refinement Tree (see Figure 8-1), the requirements engineer is able to identify the overall goal of the system, the decomposition of the system in a hierarchy of sub-organizations, roles involved in each sub-organization, and the goals that are carried out by each role in the corresponding sub-organization. The Diplomacy Game Refinement Tree provides information for the organizational, functional, and structural perspectives of the case study system. Moreover, the relevant entities of the environment of the game are identified in the Diplomacy Game Domain Model (see Figure 8-3), providing information for the organizational and structural perspective.

In addition, the social behavior needed for each role to carry out their goals is specified by mean of one social behavior diagram for each sub-organization (e.g. Figure 8-5). The case study presents a variety of social characteristics that allow to fully evaluating the proposed social behavior diagram. In particular, we identified relationships of collaboration, disposition and veracity. The social behavior diagrams provide information for the social behavior perspective. With the construction of one activity diagram for each goal, the requirements engineer is able to refine each goal in activities and protocols, and also to refine the social behavior identified in the previous activity. It is proper to mention that the collaboration relationships identified in the social behavior diagrams is refined in the Activity diagrams. As an example, the initiator protocols and reactive protocols in Figure 8-10 show the specification of the collaboration relation identified in the Social behavior diagram of Figure 8-5. The Activity diagrams also provide information for the functional and social behavior perspectives.

Furthermore, the Diplomacy game Environment Model (see Figure 8-17), identifies the resources of the system, and defines the permissions that roles have in those resources, providing information for the structural and functional perspectives. The organizational rules of the game are specified (see Table 8-1), providing information for the organizational, structural and functional perspectives.

Due to its characteristics, the Diplomacy game case study offers a good example to validate the feasibility of our approach to

model the requirements of a MAS covering its organizational, structural, functional, and social behavior properties.

Also, the use of our proposal to model the requirements of a case study with medium complexity characteristics, has served to qualitatively assess the process and the models proposed. With respect to the process, we propose a process that consists of only two main activities, in turn each activity consists of no more than three steps, and also, provides freedom to return to the previous phase if feedback is necessary. In our particular case, that freedom was useful to define the goals at a suitable level of abstraction after several iterations. Taking the refinement tree already created in the identification phase, and when trying to create the activity diagrams in the specification phase for each goal identified, we could assess whether the goals identified had an adequate level of abstraction or not, and if necessary go back to the identification phase to modified it. On the other hand, with respect to the models generated, we propose the use of simple and familiar diagrams (most of the notations are stereotyped UML standards), suitable for the identification and specification of MAS requirements. In particular, the greatest contribution is the explicit representation of social behavior, with the Social Behavior Diagram and the Activity Diagram. Having this information as a part of a model, not just a textual description, provides a step forward to validation and automation of such important characteristics in the domain.

# Chapter 9

## Conclusions

This chapter presents the main contributions of the work, future lines of work, and publications derived from the research work performed in this thesis.

### 9.1. Contributions

In recent years, various methodologies have been proposed to guide the development of multi-agent systems (MAS), such as Tropos [29], Ingenias [31], Gaia [73], etc. However, despite the importance of the requirements phase in the development of software systems, many of the proposed methodologies for the development of MAS do not adequately cover the requirements engineering phase [13], focusing mainly on the design and implementation phases. Moreover, a recent study on the application of requirements engineering techniques in the development of a multi-agent system [7] found that 79% of the current methodologies for MAS development use requirements engineering techniques which have been adapted from other paradigms (object orientation, knowledge engineering, etc.) [7].

However, these techniques and notations may not be sufficient to cover the nature of MAS, since these systems need, along with their organizational, structural, or functional properties, characteristics that are not normally necessary in conventional software systems. Therefore, the main objective of this master thesis is to propose a requirements modeling process to deal with user and software requirements emphasizing on the special aspects of MAS, satisfying the objectives established at the start of work:

- i) perform a literature review with which to investigate current techniques, methods, and methodologies to develop MAS;
- ii) perform a literature review with which to investigate current RE techniques, methods, and methodologies with particular emphasis on those techniques more appropriate to identify and specify requirements for complex and dynamic systems;
- iii) study and define the perspectives needed to adequately represent a MAS;

- iv) define the proposal for requirements modeling of multi-agent systems based on previous studies in the related topics, and covering the perspectives defined;
- v) extend the proposal towards a model-driven development approach, enabling usability evaluations on model integration with other MAS methodologies at different stages of development;
- vi) specify the development process through a process modeling language, to precisely specify how to use the proposed method and artifacts to be generated;
- vii) illustrate the feasibility of the approach by applying the requirements modeling process to the development of the strategic board Diplomacy Game [23];

This investigation work started in 2007 and from the previous work, the acceptance of the results, as well as the experience gained, is taken into account as a starting point of this master thesis.

With the starting experience, we perform a literature review on current techniques, methods, and methodologies to develop MAS, and current RE techniques, methods, and methodologies with particular emphasis on those techniques more appropriate to identify and specify requirements for complex and dynamic systems.

Then, to have a clear idea of which common and special characteristics to capture in a MAS we propose four basic perspectives for the modeling of MAS requirements: functional, structural, organizational, and social behavior. Being the organizational and social behavior perspectives the most important to the topic.

The social behavior classification presented, represent to some extent abstraction of human social behavior, and are those that differentiate agent paradigms from traditional software development. In this work, we use this classification to study the characteristics of social behavior and to propose mechanisms for the definition and specification of requirements of these types. In particular, in this work we focus on the proactiveness, collaboration, veracity, and disposition following characteristics. Social behavior is a skill that must have an agent in a MAS. Moreover, if we consider the organizational metaphor, an agent can, at different times in its life-

cycle, play one or more specific roles, which in turn have a set of responsibilities and goals. We therefore propose to identify these features of social behavior in the requirements modeling process at role level, through an analysis of the goals that need to be attained. Therefore, in the later phases of the software development, when an agent has to be defined, the corresponding roles of which a given agent will be composed will determine the agent's complete social behavior.

After having defined the MAS perspectives, we proposed a requirements modeling process for MAS. The approach is organized into two main activities: Requirements Definition and Requirements Specification. In the Requirements Definition activity the following is modeled:

- organizational structure and structural properties of the system;
- functional behavior of the system;
- domain entities and their relationships.

In the Requirements Specification activity the requirements specifications are refined, identifying:

- interactions on which the social behavior of the system is based;
- mains activities which conform the functional behavior of each role;
- permissions of the roles in the domain entities;
- structural and functional behavior.

This process supports the four perspectives that characterize a MAS: organizational, structural, functional and social behavior (proactiveness, collaboration, veracity, and disposition). We believe that this proposal addresses the need for a requirements modeling process for MAS because it incorporates specific abstractions needed to capture and specify these four perspectives. In particular, the definition and specification of features of social behavior at the requirements level will increase the quality of specifications, thus providing the expressiveness needed by the MAS in an early stage of the software development process.

We also propose an extension of the proposal for MAS requirements modeling for a model-driven development approach. We define a metamodel (level M2 of the MOF architecture) for the

requirement modeling for MAS proposal. We present a description of the metamodel and the associated graphical syntax. The metaclasses of the metamodel presented are organized in different interrelated models and represent all the information identified in the Requirements Definition and Requirements Specification phases of the approach MAS. In the context of AOSE, we have identified certain advantages of our approach. The metamodel presented defines a common set of requirements core concepts for the development of MAS through models that provides information which is necessary to cover the different perspectives of a MAS. In particular, the modeling of features of social behavior at the requirements level will increase the quality of specifications, thus providing the expressiveness needed by the MAS developer in an early stage of the software development process. In addition, the model-driven development approach provides a better means to address and solve interoperability issues, quality of reuse, maintainability, etc. We thus move towards bridging the gap between MAS requirements, captured as requirements models, and analysis and design models. The use of a model-driven development approach to develop MAS is also useful to enhance the potential, improve the quality and efficiency of AOSE, and consequently allow AOSE to be widely adopted by researchers and practitioners in the software engineering community.

The next step was to define the software development process. The quality of the development process used has a direct impact on final product quality. Therefore it is necessary a precise definition of the processes to be followed to implement the proposed method. We used SPEM as a process modeling language. SPEM is a meta-model proposed as a notation for defining software development processes and components. It was selected by the advantage of being a standard proposed by the OMG, being easy to integrate with the rest of OMG standards, and UML profiles can be utilized to benefit from the tools that support UML modeling. In the process specification, the disciplines involved in the proposed process were included; indicating for each the group of tasks, and models generated in the process. Also, was stated in the description of the process, the phases involved in the process and the order in which tasks must be applied to implement our proposal. Input and output models were also

included for each task, allowing the identification of dependencies between activities and models.

Finally, the proposal was validated using the requirements modeling of the Diplomacy Game as a case study. In particular, the social behavior characteristics and complexity of this game make them appropriate subjects for resolution with the agent-oriented paradigm. Also, the game development domain, in general, given its characteristics, particularly allows us to observe and reason about different ways in which to identify, define, and specify requirements of social behavior, in addition to the organizational (because of the various phases of which a game is composed), the structural (owing to the different types of elements used), and the functional (because of the different actions to be performed).

## 9.2. Future Work

This work opens five future research lines: improvement of the modeling requirements for MAS proposed, to cover the complete list of social behavior characteristics, definition of model transformations, CASE tool development, empirical studies to validate the proposal, and integration within a Product Line approach.

The first line of research is to include identification and specification of the characteristics of social behavior not covered in the current work: adaptability and mobility. It is also planned to perform the integration of the integration of the two characteristics in the MDSO context.

The second line of research is to complete the integration of the proposal with the MDSO approach by defining the transformations necessary so that, through the implementation of a transformation engine, the artifacts of the requirements specification can be obtained starting from the artifacts in the requirements specification. In addition, we plan to define model transformations that will allow the MAS requirements models generated to be transformed into MAS analysis and design models, and provide the infrastructure needed to generate source code from platform independent models.

The third line of research is to build a graphical editor in order to support the overall process defined, and help users to define the



different models proposed in our approach, using the Eclipse Graphical Framework [65].

The fourth line of research is to empirically validate our approach through a series of experiments using game development experts as subjects. In particular, specifying the requirements of different game applications, following the model-driven and agent-based technology as a means to validate and improve our approach.

Finally, the fifth line of research aims to integrate and extend the RE for MAS approach proposed within a development framework for Software Product Lines. Software Product Lines (SPL) is a software development approach that promotes the reuse of assets in a family of products by identifying and specifying those characteristics that are common and variable product family.

### 9.3. Related Publications

During the development of the present master thesis, different publications were accomplished. The following list gathers these publications:

- **XI Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software. CibSE 2008. (full paper published)**  
Rodríguez, L., Hume, A., Cernuzzi, L., Insfrán, E.: *Análisis Comparativo de Métodos de Elicitación de Requisitos para Sistemas Basados en Agentes*. XI Iberoamerican Workshop on Requirements Engineering and Software Environments (CibSE 2008), 253-266, 11-15 de Febrero, 2008, Recife, Pernambuco, Brasil, ISBN: 85-7084-134-5 pp. 253-266

This article presented the results of a requirements method comparison, in particular identifying a common set of MAS abstractions and some desirable qualities for the elicitation of requirements. In particular, was considered the approaches proposed by Agentis, GBRAM and RETO.

This is one of the most important conferences in Latin America in the area of Software Engineering. The official

languages are English, Portuguese and Spanish. All articles are externally peer reviewed. All the articles are of investigation, and the Scientific Committee is international composed of professors and researchers from Latin America. Articles from this conference are indexed in DBLP which gives wider dissemination.

- **9th International Conference on Quality Software. QSIC 2009. (short paper published)**

Rodriguez, L., Hume, A., Cernuzzi, L., Insfrán, E.: *Improving the Quality of Agent-Based Systems: Integration of Requirements Modeling into Gaia*. 9th International Conference on Quality Software (QSIC 2009), 278-283, August 24-25, 2009, Jeju Island, Korea, IEEE Press. ISBN: 978-0-7695-3828-0. pp. 278-283

This article presented a requirements modeling phase to extend Gaia methodology, one of the most recognized agent-oriented methodologies. The proposal includes the adoption of techniques from goal-oriented and functional-oriented approaches for the modeling of requirements. It described how these complementary proposed techniques contribute to the models provided by Gaia in its analysis and design phase establishing a clear traceability framework.

QSIC is listed in CORE conference ranking ([www.core.edu.au](http://www.core.edu.au)), as type A congress in 2007. Belongs to the Quality Software area of research, and is one of the most important conferences in the area. The language is English. All the articles are of investigation, and the Scientific Committee is international. All articles are externally peer reviewed.

- **XXXVI Conferencia Latinoamericana de Informática. CLEI 2010. (full paper published)**

Rodriguez, L., Blanes, D., Insfran, E., Cernuzzi, L.: *Requisitos de Comportamiento Social para Sistemas Multi-Agente*. XXXVI Conferencia Latinoamericana de Informática (CLEI 2010), 18-

22 de octubre, 2010, Asunción, Paraguay, ISBN: 978-99967-612-0-1

This article discussed and proposed a preliminary approach for the definition and specification of special social behaviour features of a MAS (e.g., pro-activity, adaptability, collaboration, etc.) that should be dealt with from earlier stages of software development.

This is one of the most important conferences in Latin America in the area of Software Engineering. The official languages is English, Portuguese and Spanish. All articles are externally peer reviewed. All the articles are of investigation, and the Scientific Committee is international composed of professors and researchers from Latin America, USA, and Europe. In the year 2010 had an acceptance rate of 28%.

- **Multi-Agent Systems. (Book chapter in press)**

Rodriguez, L., Insfran, E., and Cernuzzi, L.: *Requirements Modeling for Multi-Agent Systems*. In: *Multi-Agent Systems*. ISBN: 978-953-307-568-6, Intech

This book chapter presented the main contribution of this master thesis: the requirements modeling proposal emphasizing both the social behavior and the organizational aspects as key aspects for the development of MAS. Also, to illustrate the feasibility of the proposal, described the application of the requirements modeling process to the development of the strategic board game called Diplomacy.

InTech is an international publisher. The language is English. All the articles are of research, and the Scientific Committee is international composed of researchers from Europe, USA and Asia. All articles are externally peer reviewed.

- **23rd International Conference on Advanced Information Systems Engineering. CAiSE 2011. (full paper sent)**

Rodriguez, L., Insfran, E., and Cernuzzi, L., Ghose, A.: *A Software Requirements Metamodel for Multi-Agent Systems*, Sent to: 23rd International Conference on Advanced Information Systems Engineering, London, United Kingdom

This paper presents the software requirements metamodel for MAS based on the requirements modeling for the MAS approach presented as main proposal of this master thesis. The metamodel captures the organizational, structural, functional, and social behavior properties of a MAS.

CAiSE is listed in CORE conference ranking ([www.core.edu.au](http://www.core.edu.au)), as type A congress in 2008. Belongs to the Software Engineering and Information Systems area of research, and is one of the most important conferences in the area. The language is English. All articles are externally peer reviewed.

# Bibliography

- [1] Anton, A. (1996). Goal-based requirements analysis. Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96), pp.136–144, ISBN: 0-8186-7252-8, Colorado Springs, April 1996, IEEE Computer Society, Colorado
- [2] Argente, E., Botti, V., & Julián, V. (DCAI 2009). Organizational-Oriented Methodological Guidelines for Designing Virtual Organizations. International Symposium on Distributed Computing and Artificial Intelligence. 2, pp. 154-162. Salamanca, Spain: LNCS Springer
- [3] Basu, A., Hayden, M., Morrisett, G., and von Eicken, T. (1997). A Language-Based Approach to Protocol Construction. In Proc. ACM SIGPLAN Workshop on Domain Specific Languages
- [4] Bauer, B., Muller, J. P., and Odell, J. (2001). Agent UML: A formalism for specifying multiagent software systems. International Journal of Software Engineering and Knowledge Engineering, 11(3):207
- [5] Bernon, C., Cossentino, M., Gleizes, M. P., Turci, P., Zambonelli, F. (2005). A study of some multi-agent meta-models. In: Odell, J., Giorgini, P., Müller, J. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 62–67. Springer-Verlag
- [6] Beydeda, S., Book, M., & Gruhn, V. (2005). Model-Driven Software Development. New York, USA: Springer
- [7] Blanes, D.; Insfrán, E.; Abrahão, S. (2009) a. Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review. Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), pp. 510 – 517, ISBN: 0302-9743, Burgos, Spain, September 2009, Springer-Verlag, Berlin, Heidelberg
- [8] Blanes, D.; Insfrán, E.; Abrahão, S. (2009) b. RE4Gaia: A Requirements Modeling Approach for the Development of Multi-Agent Systems. International Conference on Advanced Software Engineering and Its Applications (ASEA'09), pp. 245 – 252, ISBN: 978-3-642-10618-7, Jeju Island, Korea, December 2009, Springer, Berlin
- [9] Boehm, B.W. (1979). Guidelines for verifying and validating software requirements and design specifications, EURO IFIP 79, North Holland 1979, S. 711-719
- [10] Brackett, John W. (1990). Software Requirements. SEI Curriculum Module SEI-CM-19-1.2. Boston University
- [11] Brazier, F.M.T., Dunin Keplicz, B.M., Jennings, N.R. and Treur, J. (1997). DESIRE: modelling multi-agent systems in a compositional formal framework, In M. Huhns, M. Singh, (Eds.), International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems
- [12] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gómez-Sanz, J., Pavón, J., Kerney, P., Stark, J., and Massonet, P. (2001). Agent oriented analysis using MESSAGE/UML. In Wooldridge, M., Weiß, G., and Cianciarini, P., editors, Agent-Oriented Software Engineering II: 2nd International Workshop, LNCS 2222, pages 119–135. Springer Verlag

- [13] Cernuzzi, L.; Cossentino, M.; Zambonelli, F. (2005). Process models for agent-based development. *Engineering Applications of Artificial Intelligence*, 18, 2, (March 2005) page numbers (205 - 222), ISSN: 0952-1976
- [14] Cheng, B., & Atlee, J. (2007). Research directions in Requirements Engineering. *Future of Software Engineering*,. Volume , Issue , Page(s):285 - 303
- [15] Consel, C., Hamdi, H., Réveillère, L., Singaravelu, L., Yu, H., and Pu. C. (2002). Spidle: A DSL approach to specifying streaming applications. Technical Report RR1282-02, LaBRI, Bordeaux, France
- [16] Cossentino, M. (2005). From requirements to code with the PASSI methodology. In: Henderson-Sellers, B., Giorgini, P. (Eds.) *Agent-Oriented Methodologies*, Idea Group Publishing, 79--106
- [17] Davis, A. (1996). Private communication
- [18] David A. Watt. (1990). *Programming language concepts and paradigms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. ISBN 0-13-728874-3
- [19] DeLoach, S. (2001). Analysis and Design using MaSE and agenTool. In Proc. of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS), Miami University. Miami University Press.
- [20] DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering* 11(3) , 231-258
- [21] Devanbu, P., and Poulin J., editors. *Modular Domain Specific Languages and Tools*, 1998. IEEE Computer Society Press.
- [22] ESA (2010). Entertainment Software Association, Industry Facts. Last accessed on September 14, 2010, en <http://www.theesa.com/facts/index.asp>
- [23] Fabregues, A.; Navarro D.; Serrano A.; Sierra C. (2010). dipGame: a Testbed for Multiagent Systems, *Proceeding of the ninth International Conference of Autonomous Agents and Multi-Agent Systems*, Toronto, Canada, May 2010
- [24] Falkenberg, E.D.; Hesse, W.; Lindgreen, P.; Nilsson, B. E.; Oei, J.L.H.; Rolland, C.; Stamper, R. K.; Van Assche, F.J.M.; Verrijn-Stuart, A. A.; Voss., K. (1998). *FRISCO - A framework of information system concepts - The FRISCO Report*. Task Group FRISCO, ISBN: 3-901882-01-4
- [25] Ferber, J., Gutknecht, O., & Michel, F. (2004). From Agents to Organizations: An Organizational View of Multi-Agent Systems. In P. Giorgini, J. Müller, J. Odell (Eds.), *Agent-Oriented Software Engineering (AOSE) IV* (pp.214-230). LNCS 2935, Springer
- [26] Franklin, S. and A. Gasser. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. In Muller, Wooldridge, and Jennings, eds. *Intelligent Agents III. Agent Theories, Architectures, and Languages*. Springer Verlag, 1997. p.21-35
- [27] Fuggetta, A. (2000). Software process: a roadmap. *International Conference on Software Engineering, Future of Software Engineering Track* (pp. 25-34). Limerick, Ireland: ACM
- [28] Gerber, A., and Raymond, K. MOF to EMF: there and back again. In *eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 60-64, New York, NY, USA, 2003. ACM Press.

- [29] Giorgini, P.; Kolp, M.; Mylopoulos, J.; Castro, J. (2005). Tropos: A Requirements-Driven Methodology for Agent-Oriented Software. In: Agent-Oriented Methodologies, Brian Henderson-Sellers; Paolo Giorgini, page numbers (20-45), Idea Group , ISBN: 1591405815, USA
- [30] Giret, A., Botti, V. J., & Valero, S. (2005). MAS Methodology for HMS. Holonic and Multi-Agent Systems for Manufacturing, Second International Conference on Industrial Applications, of Holonic and Multi-Agent Systems, HoloMAS 2005 (pp. 39-49). Copenhagen, Denmark: Lecture Notes in Computer Science
- [31] Gómez-Sanz, J.; Pavón, J. (2003). Agent Oriented Software Engineering with INGENIAS, Proceedings of the 3rd Central and Eastern Europe Conference on Multiagent Systems (CEEMAS '03), pp. 394 - 403, ISBN: 3-540-40450-3, Prague, June 2003, Springer-Verlag, Prague
- [32] Hector, A.; Lakshmi Narasimhan, V. (2005). A New Classification Scheme for Software Agents. Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05), pp. 191 - 196, ISBN: 0-7695-2316-1, Sydney, Australia, July 2005, IEEE Computer Society, Washington, DC
- [33] Huzam S. F. Al-Subaie; Maibaum Tom S. E. (2006). Evaluating the Effectiveness of a Goal-Oriented Requirements Engineering Method. Proceedings of the Fourth International Workshop on Comparative Evaluation in Requirements Engineering, pp. 8 - 19, ISBN: 0-7695-2712-4, Minneapolis/St. Paul, Minnesota, September 2006, IEEE Computer Society Washington, DC
- [34] IEEE. IEEE Standard Glossary of Software Engineering Terminology. New York: IEEE. ANSI/IEEE Std 729-1983
- [35] Jacobson, I., Rumbaugh, J., and Booch, G. (1999). The Unified Software Development Process. Addison-Wesley
- [36] Jean Bézivin. (2005). On the unification power of models. *Software and Systems Modeling*, 4(2):171-188
- [37] Jean Bézivin and Olivier Gerbé. (2001). Towards a Precise Definition of the OMG/MDA Framework. In ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering, page 273, Washington, DC, USA, 2001. IEEE Computer Society
- [38] Jean M. Favre. (2004). Towards a Basic Theory to Model Model Driven Engineering. Workshop on Software Model Engineering
- [39] Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM*, 44, 4 (Apr.), 35-41
- [40] Jitnah, D., Han, J., Steele, P. (1995). Software Requirements Engineering: An Overview, Technical Report 95-04, Peninsula School of Computing and Information Technology, Monash University, Melbourne, Australia
- [41] Jouault, F., and Bézivin, J. (2006). KM3: a DSL for Metamodel Specification. Pages 171-185
- [42] Kleppe A., Warmer, J., and Bast, W. (2003). MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley
- [43] Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., and

- Schwinger, W. (2006). Towards a semantic infrastructure supporting model-based tool integration. In *GaMMa '06: Proceedings of the 2006 international workshop on Global integrated model management*, pages 43–46, New York, NY, USA
- [44] Leite, J.C.S.P. (1987). A survey on requirements analysis. Technical Report RTP-071, Department of Information and Computer Science, University of California at Irvine, Irvine, CA 92717
- [45] Lind, J. (2001). Iterative Software Engineering for Multiagent Systems, the MASSIVE Method
- [46] Maiden, N. (1998). CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, 5, 4, (October 1998) page numbers (419 - 446), ISSN: 0928-8910
- [47] Mellor, S. J., Scott, K., Uhl, A., Weise, D. (2004). *MDA Distilled*, Addison-Wesley
- [48] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, ISSN 0360-0300. URL <http://portal.acm.org/citation.cfm?id=1118890.1118892>
- [49] Moraitis, P., Spanoudakis, N. I. (2006). The Gaia2Jade process for multi-agent systems development. *J. Appl. Artif. Intell.* 20, 251–273
- [50] Muller Pierre-Alain. (2005). Weaving Executability into Object-Oriented Meta-Languages. In *MODELS/UML'2005*, pages 264–278. Springer
- [51] Nuseibeh, B.A. and Easterbrook, S.M. (2000). Requirements Engineering: A Roadmap. In A. C.W. Finkelstein (ed) "The Future of Software Engineering". (Companion volume to the Proc. of the 22nd Int. Conf. on Software Engineering, ICSE00) IEEE Computer Society Press, pp. 35-46.
- [52] Odell, J.; Parunak, H. V. D.; Bauer, B. (2000). Extending UML for agents. *Proceeding of the 2nd Int. Workshop on Agent-Oriented Information Systems*, pp. 3 – 17, Berlin, iCue Publishing
- [53] O'Hare, G. M. P. and Jennings, N. R. (1996), *Foundations of distributed artificial intelligence*, John Wiley & Sons, ISBN 0-471-49691-X
- [54] Object Management Group. (2009, 11 1). Retrieved 11 1, 2010, from <http://www.omg.org/>
- [55] Object Management Group. (2004). *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group, Inc.
- [56] OMG (Object Management Group). *Software Process Engineering Meta-Model (SPEM), version 2.0*. Last accessed on January, 2011, in <http://www.omg.org/spec/SPEM/2.0/PDF/>
- [57] Pavón, J., Gómez-Sanz, J. J., Fuentes, R. (2006). Model driven development of multi-agent systems. In Rensink, A., Warmer, J. (Eds.), *ECMDA-FA 2006*. LNCS, vol. 4066, pp. 284--298. Springer
- [58] Penserini, L., Perini, A., Susi, A. and Mylopoulos, J. (2005). *From Stakeholder Intentions to Agent Capabilities*. Technical report, ITC-irst, Trento, Italy
- [59] Picard, G., Gleizes, M. P. (2004). The ADELFE methodology. In: *Methodologies and Software Engineering for Agent Systems, The Agent-oriented Software Engineering Handbook*. Kluwer Academic Publishers



- [60] Rodriguez L.; Hume, A.; Cernuzzi, L.; Insfrán, E. (2009). Improving the Quality of Agent-Based Systems: Integration of Requirements Modeling into Gaia. Proceedings of the Ninth International Conference on Quality Software (QSIC '09), pp. 278 - 283, ISBN: 978-0-7695-3828-0, Jeju, Korea, August 2009, IEEE Computer Society Washington, DC
- [61] Russell, S.J. (1997). Rationality and intelligence. *Artificial Intelligence*, Vol. 94, 1997. p.57-77
- [62] Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, Vol. 60, 1993. p.51-92
- [63] Sommerville, I. (2002). *Ingeniería de Software*. Pearson Educación
- [64] Southwell, K., J. James, et al. (1987). *Requirements Definition and Design. The STARTS Guide, Second Edition, Vol. 1*, National Computing Center.
- [65] The Eclipse Foundation. Last accessed on September 2010, from <http://www.eclipse.org>
- [66] Thibault, S., Marlet, R., and Consel, C. (1999). Domain-Specific Languages: From Design to Implementation Application to Video Device Drivers Generation. *Software Engineering*, 25(3):363-377
- [67] van Deursen, A., and Klint, P. (1997). Little languages: little maintenance? In 62, page 17. *Centrum voor Wiskunde en Informatica (CWI)*, ISSN 1386-369X, 30
- [68] Van Lamsweerde, A.; Darimont, R.; Letier, E. (1998). Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24, 11, (November 1998) page numbers (908 - 926), ISSN: 0098-5589
- [69] Wizards (2010). The rules of Diplomacy. The game of international intrigue. Last accessed on September 14, 2010 [http://www.wizards.com/avalonhill/rules/diplomacy\\_rulebook.pdf](http://www.wizards.com/avalonhill/rules/diplomacy_rulebook.pdf)
- [70] Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowl. Eng. Rev.* 10, 2, 115-152
- [71] Wooldridge M. (2002), *An Introduction to Multi-Agent Systems*, John Wiley & Sons
- [72] Yu, E. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE '97), pp. 226 - 235, ISBN: 0-8186-7740-6, Washington D.C., USA, January 1997
- [73] Zambonelli, F.; Jennings, N.; Wooldridge, M. (2003). Developing Multiagent Systems: The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12, 3, (July 2003) page numbers (317 - 370), ISSN: 1049-331X
- [74] Zave, P. (1997) Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, 29(4): 315-32