

Universidade Federal de Pernambuco
Centro de Informática

Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case

Carla Taciana Lima Lourenço Silva

Recife,
April 2007

Carla Taciana Lima Lourenço Silva

**Separating Crosscutting Concerns in Agent Oriented
Detailed Design: The Social Patterns Case**

Doctoral Thesis Proposal

*Thesis presented to the Graduate
Program in Computer Science of the
Universidade Federal de Pernambuco
in partial fulfillment of the
requirements for the degree of Doctor
of Science.*

Advisor: Jaelson Freire Brelaz de
Castro

Co-Advisors: João Baptista da Silva
Araújo Júnior and Patrícia Cabral de
Azevedo Restelli Tedesco

Recife,
April 2007

Silva, Carla Taciana Lima Lourenço

Separating crosscutting concerns in agent oriented detailed design: the social patterns case / Carla Taciana Lima Lourenço Silva. – Recife: O autor, 2007.

xii, 140 folhas: il., fig., tab.

Tese (doutorado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2007.

Inclui bibliografia e apêndice.

1. Engenharia de software. 2. Desenvolvimento orientado a agentes. 3. Desenvolvimento orientado a aspectos. 4. Projeto de Software. I. Título.

005.1

CDD (22.ed.)

MEI2007-046

Acknowledgements

Agradeço a Deus pelo apoio espiritual em todos os momentos e por permitir a realização deste sonho.

À Salete, minha mãe, meus agradecimentos pela paciência, por compreender qual o meu caminho de realização e pelas eternas palavras de incentivo. Conseguimos mais esta conquista!

Ao meu esposo Hermann, por me apoiar e incentivar em toda e qualquer situação.

A Jaelson, meus agradecimentos por acreditar no meu potencial e pelo suporte acadêmico e emocional durante este período que me transformou na pessoa que sou hoje. Sem sua orientação eu não teria chegado até aqui!

A João Araújo e Ana Moreira, minhas saudades e agradecimentos pela assistência incondicional e orientação contínua apesar da distância.

À Patrícia Tedesco, meus agradecimentos pelo contínuo incentivo e pela colaboração para que este trabalho se concretizasse.

Aos meus amigos, meus agradecimentos pela amizade, paciência e compreensão nos momentos de ausência e pelos inesquecíveis momentos de descontração.

Abstract

Although agent-oriented software engineering is quite recent, this new paradigm has been successfully used in industrial applications, such as telecommunications and e-commerce. Among the several issues required for the consolidation of this new paradigm, we highlight the need for suitable methodologies, notations and tools to support agent-oriented software development. In this light, the Tropos framework proposes a requirements-driven approach aimed at building agent-oriented software that operates within a dynamic environment. Among the challenges of multi-agent systems (MAS) development, we highlight the issue of developing reusable, flexible, understandable and maintainable systems in order to reduce development costs. To this end, Tropos has defined a set of design patterns, called social patterns, focusing on social and intentional features which are recurrent in cooperative and multi-agent systems. These patterns can be called crosscutting concerns, since they cut across the functional modules in MAS, decreasing the system reusability and maintainability. These crosscutting concerns can be better addressed by adopting aspect-oriented software development techniques. The purpose of this new technology is to encapsulate crosscutting concerns in separate modules, the aspects, avoiding their scattering or tangling through software artifacts. Thus, aspects can be used as abstractions to capture social patterns concerns. This thesis proposes a description technique to promote advanced separation of social patterns concerns in MAS design. To achieve this we have: (i) specified a UML (Unified Modeling Language) profile that defines a metamodel incorporating agency concepts; (ii) specialized the agency metamodel by using the concept of model roles for specifying which model elements must participate in a pattern; (iii) attached notes to the structural model of the social pattern to capture some abstractions of aspect orientation; and (iv) suggested guidelines to map the proposed notation to the constructs of AspectJ and JADE implementation environments. In particular, we have used an integration of JADE and

AspectJ to implement our approach. Moreover, we have described a process, using the SPEM (Software Process Engineering Metamodel) notation, to guide the whole detailed design of a MAS using the social patterns. To illustrate the feasibility of our proposal, we specify two multi-agent systems from different application domains and with different degrees of complexity: a Content Manager System, called e-News, and a Conference Management System.

Keywords: Multi-Agent Systems, Design Patterns, Aspect Orientation, Modeling, Detailed Design Process.

Resumo

O desenvolvimento orientado a agentes é bastante recente, no entanto, este novo paradigma tem sido utilizado cada vez mais em aplicações industriais, tais como telecomunicações e comércio eletrônico. Entre as principais preocupações para a consolidação deste novo paradigma, destacamos a necessidade de técnicas, notações e ferramentas adequadas para suportar o desenvolvimento de sistemas orientados a agentes. Neste sentido, o projeto Tropos está desenvolvendo uma abordagem de desenvolvimento orientado a agentes centrada em requisitos que visa construir software orientado a agentes que opera dentro de um ambiente dinâmico. Dentre os desafios do desenvolvimento de sistemas multi-agentes (SMA), destaca-se a questão de desenvolver sistemas reusáveis, flexíveis, fáceis de entender e manter de forma a reduzir os custos de desenvolvimento. Para este fim, Tropos definiu um conjunto de padrões de projeto, chamados padrões sociais, focando em aspectos intencionais e sociais que são recorrentes em sistemas multi-agentes e cooperativos. Estes padrões podem ser chamados de interesses entrecortantes (do inglês, *crosscutting concerns*), já que eles entrecortam módulos funcionais em SMA, diminuindo a reusabilidade e manutenibilidade do sistema. Estes interesses entrecortantes podem ser mais bem endereçados pela adoção de técnicas do desenvolvimento de software orientado a aspectos. O objetivo desta nova tecnologia é encapsular interesses entrecortantes em módulos separados, os aspectos, evitando seu espalhamento e entrelaçamento através dos artefatos de software. Assim, aspectos podem ser usados como abstrações para capturar interesses de padrões sociais. Esta tese propõe uma técnica de descrição que visa promover a separação avançada de interesses de padrões sociais no projeto de SMA. Para alcançar este fim, nós: (i) especificamos um *profile* UML (*Unified Modeling Language*) que define um metamodelo que incorpora conceitos de agência; (ii) especializamos o metamodelo de agência usando o conceito de papéis de modelo (do inglês, *model roles*) para especificar quais elementos do modelo

devem participar em um padrão; (iii) anexamos notas ao modelo estrutural do padrão social para capturar algumas abstrações da orientação a aspectos; e (iv) sugerimos diretrizes para mapear a notação proposta em construtores dos ambientes de implementação AspectJ e JADE. Em particular, nós usamos uma integração de JADE e AspectJ para implementar nossa abordagem. Além disso, nós descrevemos um processo, usando a notação SPEM (*Software Process Engineering Metamodel*), para guiar o projeto detalhado completo de um SMA usando os padrões sociais. Para ilustrar a viabilidade de nossa proposta, nós especificamos dois sistemas multi-agentes de diferentes domínios de aplicação e com diferentes graus de complexidade: um Sistema Gestor de Conteúdo, chamado e-News, e um Sistema Gestor de Conferência.

Palavras-Chaves: Sistemas Multi-Agentes, Padrões de Projeto, Orientação a Aspectos, Modelagem, Processo de Projeto Detalhado.

Abbreviations

ACL - Agent Communication Language

ADL - Architectural Description Language

AID - Agent Id

AODM - Aspect-Oriented Design Model

AOSD - aspect-oriented software development

API - Application Programming Interface

APS - Agent Pattern Specifications

ARKnowD - Agent-oriented Recipe for Knowledge Management Systems Development

AsoC - Advanced Separation of Concerns

BDI - Belief-Desire-Intention)

CAPS - Communication Agent Pattern Specification

CASE - Computer Aided Software Engineering

DF - Directory Facilitator

EC - Expert Committee

FIPA - Foundation for Intelligent Physical Agents

IPS - Interaction Pattern Specification

MAS - Multi-Agent Systems

OCL - Object Constraint Language

PSs - Pattern Specifications

SAPS - Structural Agent Pattern Specification

SD - Strategic Dependency

SPEM - Software Process Engineering Metamodel

SR - Strategic Rationale

TAO - Taming Agents and Objects

UFO - Unified Foundational Ontology

UML - Unified Modeling Language

Table of Contents

Abstract.....	II
Resumo	IV
Abbreviations	VI
1 Introduction	1
1.1 Current Stage of Agent-Oriented Software Engineering	2
1.2 The Problem	3
1.3 The Proposed Solution	5
1.4 Thesis Outline	6
2 Agent-Oriented Software Engineering	8
2.1 Agent-Oriented Paradigm	10
2.2 Tropos	12
2.3 Agent-Oriented Design Patterns	18
2.4 Summary	23
3 Aspect-Oriented Software Development	25
3.1 Aspect-Oriented Paradigm	28
3.2 Aspect-Oriented Modeling in Software Design	30
3.3 Summary	38
4 Social Patterns in Tropos	40
4.1 Social Patterns Description	41
4.2 The Crosscutting Nature of the Matchmaker Pattern	48
4.3 The Detailed Design Phase in Tropos.....	50
4.4 Summary	51
5 Agency Metamodel and Notation.....	53
5.1 An Agency Metamodel	54
5.2 Agent-Oriented Modeling	60
5.3 Comparison with Related Work	66
5.4 Summary	69
6 Agent Pattern Specifications	70
6.1 A Technique for Social Pattern Specification	72
6.2 Towards Implementation in JADE and AspectJ.....	78
6.3 Comparison with Related Work	79
6.4 Summary	81
7 Detailed Design Process	83
7.1 Detailed Design Process in SPEM.....	84
7.2 Comparison with Related Work	95

7.3	Summary	95
8	Case Study: Newspaper Office.....	97
8.1	Detailed Design	98
8.2	Agent Detailed Design	100
8.3	Social Patterns Selection and Application	111
8.4	Summary	116
9	Conclusions	117
9.1	Final Considerations	118
9.2	Comparative Analysis	118
9.3	Contributions	121
9.4	Future Work.....	122

List of Figures

<i>Figure 2.1 – Strategic Dependency Model for Newspaper Office</i>	15
<i>Figure 2.2 – Strategic Rationale model for Editor in Chief actor</i>	17
<i>Figure 3.1 – Example of crosscutting concern scattering and tangling into classes' hierarchy [Sousa, 2004]</i>	27
<i>Figure 3.2 – Illustration of the separation and composition of aspects [Sousa, 2004]</i>	28
<i>Figure 3.3 - Design-level aspects and crosscutting interfaces in aSideML.</i>	32
<i>Figure 3.4 - Crosscutting Interfaces and Architecture-Level Aspects.</i>	32
<i>Figure 3.5 - An SPS and a structurally conforming class diagram. (a) A conforming class diagram and (b) the SPS [France et al., 2004]</i>	34
<i>Figure 3.6 - An IPS (a) and a Conforming Sequence Diagram (b) [France et al., 2004]</i>	35
<i>Figure 3.7 - The Aspect-Oriented Architecture of EC Agents [Chavez et al., 2005]</i>	37
<i>Figure 3.8 - Refining the Architectural Aspects and Crosscutting Interfaces [Chavez et al., 2005]</i>	38
<i>Figure 4.1 – Social Diagram for the Matchmaker Pattern [Kolp et al., 2005]</i>	42
<i>Figure 4.2 – Structural Diagram Template [Kolp et al., 2005]</i>	44
<i>Figure 4.3 – Communication diagram for the Matchmaker pattern [Kolp et al., 2005]</i>	46
<i>Figure 4.4 – Dynamic Diagram – Matchmaker</i>	47
<i>Figure 4.5 – Code to register the service in the yellow pages</i>	49
<i>Figure 4.6 – Code to unregister the service in the yellow pages</i>	49
<i>Figure 4.7 – Code to search in the yellow pages for the agent which provides a service</i>	49
<i>Figure 5.1 – Agency metamodel reflecting intentional concepts</i>	56
<i>Figure 5.2 – Agency metamodel reflecting interaction concepts</i>	58
<i>Figure 5.3 – Agency stereotypes</i>	60
<i>Figure 5.4 – MAS Architectural Diagram</i>	61
<i>Figure 5.5 – MAS Communication Diagram</i>	62
<i>Figure 5.6 – MAS Environmental Diagram</i>	63
<i>Figure 5.7 – MAS Intentional Diagram</i>	64
<i>Figure 5.8 – MAS Rational Diagram</i>	64
<i>Figure 5.9 – MAS Plan Diagram</i>	66
<i>Figure 6.1 – Matchmaker Structural Agent Pattern Specification</i>	76
<i>Figure 6.2 – Matchmaker Communication Agent Pattern Specification</i>	78
<i>Figure 7.1 – Detailed Design Package</i>	85
<i>Figure 7.2 – WorkDefinitions Flow</i>	87
<i>Figure 7.3 – Workflow for the Detailed Design Process</i>	88
<i>Figure 8.1 – e-News System Architecture</i>	99
<i>Figure 8.2 – Means-end analysis for Editor</i>	100
<i>Figure 8.3 – Means-end analysis for Chief Editor</i>	101
<i>Figure 8.4 – Means-end analysis for Webmaster</i>	101

<i>Figure 8.5 – Means-end analysis for Photographer-Reporter</i>	102
<i>Figure 8.6 – Architectural Diagram for e-News</i>	104
<i>Figure 8.7 – e-News Communication Diagram</i>	105
<i>Figure 8.8 – e-News Environmental Diagram</i>	106
<i>Figure 8.9 – e-News Intentional Diagram</i>	108
<i>Figure 8.10 – e-News Rational Diagram</i>	109
<i>Figure 8.11 – Edit and Publish Newspaper Plan Diagram</i>	110
<i>Figure 8.12 – MAS Architectural Diagram weaved with the SAPS</i>	113
<i>Figure 8.13 – MAS Architectural Diagram weaved with the CAPS</i>	114
<i>Figure 8.14 – AspectJ Code for RegisterServices Aspect</i>	115
<i>Figure 8.15 – AspectJ Code for RequestProviderIdentification Aspect</i>	115
<i>Figure 8.16 – JADE Code for Photographer-Reporter Agent</i>	116
<i>Figure 8.17 – JADE Code for Editor Agent</i>	116
<i>Figure 1 – Strategic Dependency Model for the e-News System</i>	142
<i>Figure 2 – Strategic Rationale Model for the e-News System</i>	143
<i>Figure 3 – Assignment of the e-News System Responsibilities</i>	146
<i>Figure 4 – Embassy Structural Agent Pattern Specification</i>	156
<i>Figure 5 – Embassy Communication Agent Pattern Specification</i>	157
<i>Figure 6 – Mediator Structural Agent Pattern Specification</i>	159
<i>Figure 7 – Mediator Communication Agent Pattern Specification</i>	159
<i>Figure 8 – Monitor Structural Agent Pattern Specification</i>	162
<i>Figure 9 – Monitor Communication Agent Pattern Specification</i>	163
<i>Figure 10 – Wrapper Structural Agent Pattern Specification</i>	165
<i>Figure 11 – Wrapper Communication Agent Pattern Specification</i>	165
<i>Figure 12 – Broker Structural Agent Pattern Specification</i>	168
<i>Figure 13 – Broker Communication Agent Pattern Specification</i>	169

List of Tables

<i>Table 6.1 – The Template for a Pattern Description</i>	<i>72</i>
<i>Table 1 – Evaluation of Correlation Catalogue</i>	<i>145</i>
<i>Table 2 – The Template for Embassy Pattern</i>	<i>154</i>
<i>Table 3 – The Template for Mediator Pattern</i>	<i>157</i>
<i>Table 4 – The Template for Monitor Pattern</i>	<i>160</i>
<i>Table 5 – The Template for Wrapper Pattern</i>	<i>163</i>
<i>Table 6 – The Template for Broker Pattern</i>	<i>166</i>

1 Introduction

This chapter presents the main motivations of the thesis. Afterwards, it describes the current stage of agent oriented software engineering. The subsequent sections discuss the work scope, the objectives and proposed approach, as well as the document structure.

Currently, most software systems are required to operate in complex, distributed, large, open, dynamic, unpredictable, heterogeneous and highly interactive application environments [Weiss, 2000]. These emerging complexities make the development of such systems a very difficult task. Therefore, software engineers are trying to understand the characteristics of multifaceted software systems, recognizing the importance of interaction. In fact, many researchers are investigating new paradigms to increase the ability for modeling, designing and implementing complex software systems [Wooldridge, 2002].

One of the most promising software paradigms for complex systems engineering is agent orientation, since it offers a higher level of abstraction when thinking about software systems features and behavior. An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [Wooldridge, 2002]. We are not advocating agent paradigm as a *Silver Bullet*, since there is no evidence suggesting the improvement of productivity in software engineering. However, the usage of this technology in industry has demonstrated that agent oriented techniques lead to improvement of distributed complex system development [Luck et al., 2003].

1.1 Current Stage of Agent-Oriented Software Engineering

Agent technology is believed to have the potential for becoming a mainstream solution of Software Engineering (as did object orientation). However, there is no consensus about agent definition. Consequently, a set of different notations covering different agent features has been proposed [Kolp et al., 2005] [Odell et al., 2000] [Silva, V. et al., 2005]. Moreover, the benefits promised by the agent paradigm cannot be fully achieved yet because of the lack of suitable methodologies which enable designers to clearly specify and structure its applications as agent-oriented systems. Moreover, understanding the appropriate situations to apply agent-based solutions is also important [Jennings, 2003]. There is still a lack of tool support available in industry to aid agent-oriented development. Another problem with the acceptance of agent technology is a social one. Both individuals and organizations will need to become more acquainted with notion of autonomous software agents, which can act in their behalf [Jennings, 2000].

In this context, proposing development methodologies became a promising area in Software Engineering research. In fact, the increasing interest in software agents and multi-agent systems (MAS) has recently led to the development of new methodologies based on agent concepts. Methodologies such as GAIA [Zambonelli et al., 2003], AUML [Odell et al., 2000], and Tropos [Castro et al., 2002] have become the focus of the emerging area of agent-oriented software engineering. These methodologies propose different approaches for using agent concepts and techniques at various stages during the software development life cycle.

In particular, we are focusing on Tropos [Castro et al., 2002] [Giorgini et al., 2005], a requirements-driven framework aimed at building agent-oriented software that operates within a dynamic environment. In order to promote an efficient development of agent-oriented systems, Tropos supports five phases of software development: early requirements, late requirements, architectural design, detailed design and implementation.

1.2 The Problem

Among the challenges of multi-agent systems development, we highlight the issue of reusing MAS design solutions in order to reduce development costs. To this end, multi-agent systems development approaches should investigate the use of design patterns, as Tahara et al. (1999), Kendall et al. (1998), Aridor and Lange (1998) and Hayden and Yang (1999) argue. A software pattern describes a recurring problem and solution; it may address conceptual, architectural or design problems [Kendall et al., 1998]. In fact, a lot of work has been done in software engineering in order to document software design patterns as, for example, the proposal of Gamma et al. (1994). Unfortunately, they focus on object orientation and not on the inherent intentional and social characteristics of agents. To overcome this problem, Tropos has defined a set of design patterns, called social patterns [Kolp et al., 2002] [Kolp et al., 2005], focusing on social and intentional features which are recurrent in cooperative and multi-agent systems. However, Tropos does not offer a detailed description of the social patterns. In fact, there isn't a proper notation to support the design of MAS in Tropos. This makes difficult describing the social patterns in detail. Moreover, Tropos does not provide a systematic way to choose and apply the social patterns in order to properly refine the architectural components of the software under development. Indeed, the current

detailed design process should be more detailed and systematic to clearly guide software engineers in the design of MAS.

Reusing MAS design and implementation isn't guaranteed with the use of design patterns. Indeed, the current use of patterns can lead to the scattering of design pattern concerns¹. Consequently, tangling with the functional modules concerns of the application (i.e. the agent roles concerns) compromises the quality requirements aforementioned. Note that the concern related to application core (responsible to realize application core functionality) is pattern independent and can be reused. The concern related to design patterns describes the abstract structure and behavior of the pattern, which are application core independent [Noda and Kishi, 2001] [Hannemann and Kiczales, 2002] [Sant'Anna et al., 2004] [Cacho et al., 2006]. According to Noda and Kishi (2001), current software development paradigms cannot use design patterns properly, since the application core become dependent on patterns, decreasing possibilities of reusing the application functional core. However, we can use aspect orientation [Kiczales et al., 1997] to solve this problem.

Aspect-oriented software development [Kiczales et al., 1997] is an evolving paradigm to modularize concerns that other abstraction (objects/agents) are not able to capture explicitly. It encourages modular descriptions of complex software by providing support for cleanly separating the system functionality from its crosscutting concerns.

In fact, several approaches have already been proposed which use aspect orientation in design pattern design and implementation, in order to separate pattern concerns from application functional concerns [Noda and Kishi, 2001] [Hannemann and Kiczales, 2002] [Camargo et al., 2003] [Sant'Anna et al., 2004] [Cacho et al., 2006]. This evidence has motivated us to use aspect orientation in the context of Tropos framework in order to promote separation (and modularization) of social patterns concerns (abstract structure and behaviour) from agent roles concerns (application functional modules) and its later composition with the application agent roles during the detailed design and implementation of multi-agent systems.

¹ A *concern* is some part of the problem that we want to treat as a single conceptual unit in the software solution [Dijkstra, 1976].

1.3 The Proposed Solution

This work proposes an agency metamodel [Silva, C. et al., 2006c], by extending the UML metamodel [OMG, 2005b] to capture the features of agency at MAS design level. Based on this agency metamodel, we define a MAS modeling notation to support the specification of six views of MAS detailed design: architectural, communication, environmental, intentional, rational and plan. Moreover, a description technique for social patterns is provided and includes a subset of GoF's template [Gamma et al., 1994] and two UML-based diagrams to represent structural and communicational views of the pattern. In particular, this description technique promotes separation and modularization of social patterns concerns in MAS design by using abstractions and mechanisms of aspect oriented paradigm. The advanced separation of social pattern concerns is not currently taken into account in the current agent oriented software engineering.

We also define and formalize a detailed design process using the SPEM notation [OMG, 2006]. This process guides the use of the MAS modeling notation and supports selection and application of social patterns. To select the proper social patterns the developer must understand a specific design problem of an application and analyze each social pattern description to find one or more patterns which can solve the application problem. The process for using the chosen social patterns corresponds to an aspect weaving process, where the social patterns concerns are combined with the agent roles composing the system.

Aiming at closing the gap between detailed design and implementation phases of Tropos, we provide some guidelines to map our notation to constructs of JADE [Bellifemine et al., 2003] and AspectJ [Kiczales et al., 2001] implementation environments.

To illustrate the feasibility of our proposal, we have selected two systems from different application domains and with different degrees of complexity: a Content Manager System [Silva, I., 2005], called e-News [Silva, C. et al., 2005], and a Conference Management System [Zambonelli et al., 2003] [Garcia, 2004].

1.4 Thesis Outline

Besides of this introductory chapter, this work includes eight other chapters as follows:

Chapter 2 – Agent Oriented Software Engineering

This chapter discusses the current state of the art for engineering agent-oriented software systems. Currently, there is an ever-increasing demand for software systems to operate in increasingly complex and dynamic application environments. Hence, agent-orientation appears as a well-suited paradigm for building such systems, since it provides a higher level of abstraction to deal with complex software systems features and behavior.

Chapter 3 – Aspect Oriented Software Development

This chapter discusses the current state of the art for engineering aspect-oriented software systems. Crosscutting concerns are properties that do not align well with the decomposition criteria of the chosen approach and, therefore, cannot be modularized. These crosscutting concerns can be better addressed by adopting aspect-oriented software development techniques. The purpose of this new technology is to localize crosscutting concerns, avoiding their scattering or tangling through software artifacts.

Chapter 4 – Social Patterns in Tropos

In this chapter we present the current state of art on social patterns. In particular, the lack of a detailed description of social patterns is discussed. Moreover, we illustrate the crosscutting nature of a specific social pattern and point out the limitations of the current detailed design phase of Tropos.

Chapter 5 – Agency Metamodel and Notation

Agents provide developers with a flexible way to structure systems around autonomous, communicating elements. To support the efficient development of such systems, design techniques need to be introduced. In this context, we propose an extension of the UML 2.0 metamodel to support agency feature and UML-based diagrams which can be used to capture six views of multi-agent systems architecture (Architectural, Intentional, Environmental, Communication, Rational and Plan).

Chapter 6 – Agent Pattern Specifications

This chapter proposes an approach to support separation and modularization of crosscutting concerns in multi-agent systems. Aspect-Oriented Software Development offers mechanisms to encapsulate crosscutting concerns in separate modules, the *aspects*. Aspects are used as abstractions to capture social patterns concerns that cut across functional modules in MAS. To achieve this, we propose a technique to describe social patterns in an aspect-oriented context.

Chapter 7 – Detailed Design Process

In this chapter we outline a process to perform the detailed design of multi-agent systems. This process guides the description of MAS using a UML-based notation, helps the selection of the proper social pattern(s) and provides a systematic way to apply social patterns. The Detailed Design Process is described using the Software Process Engineering Metamodel (SPEM) notation.

Chapter 8 – Case Study

This chapter provides a case study to illustrate the feasibility of our proposal. In particular, we specify a multi-agent system for a Newspaper Content Management domain.

Chapter 9 – Conclusions

This chapter presents the current results of our approach, the lessons learned during this work, as well as it points out directions to future work to improve our proposal.

2 Agent-Oriented Software Engineering

Currently, there is an ever-increasing demand for software systems to operate in increasingly complex and dynamic application environments. Hence, agent-orientation appears as a well-suited paradigm for building such systems, since it provides a higher level of abstraction to deal with complex software systems features and behavior. This chapter discusses the current state of the art for engineering agent-oriented software systems.

Designing and implementing high quality industrial software is difficult. In fact, developing such projects is assumed to be amongst the most complex building tasks performed by humans. Indeed, to alleviate this challenge, several implementation paradigms have been proposed in literature (ex. structured, declarative, object oriented and component based programming). This incremental evolution of paradigms aims to make software engineering more manageable as well as to cope with the increasing of applications complexity.

From the user point of view, quality software [ISO 9126, 1996] satisfies its needs and is built in predefined budget and time. According to a software engineer point of view, quality software presents, for example, the following characteristics:

- *Flexibility* – allows to satisfy new business requirements (functionality) in an easy and quick way;
- *Adaptability* – allows to personalise an application for several users, by using several alternatives to offer the application services with a minimum impact in its kernel;
- *Maintainability* – allows to change parts of an application with a minimum impact in the other parts;
- *Reusability* – allows to build unique and dynamic applications rapidly;
- *Legacy* – allows to reuse the functionality of legacy systems into new software applications;
- *Interoperability* – allows two applications running in different platform to exchange information;
- *Scalability* - how easy a system or component can be modified to fit the problem area;
- *Robustness* – allows implementing software solutions with fewer defects.

Moreover, the designer considers a good software development process the one that presents the following features:

- *Faster development* – allows developing new systems more rapidly and with low budget.
- *Lower risk* – allows all features to be addressed to obtain a quality software, without the risk of producing failed projects.

Although object orientation is very popular in the world of software development, it does not solve all the problems. Object orientation does not have all the apparatus needed for developing in an easier way software which meets previously listed attributes. This means that the benefits offered by object orientation are not automatic, since they depend on the correct use of this technology. Moreover, many extensions of object orientation need to be made to accomplish with less effort the characteristics required for quality software. For example, object orientation extensions to support distributed systems [SUN, 2005], to better deal with crosscutting features [Kiczales et al., 1997], to develop real-time systems [Bihari et al., 1989] and so on.

Therefore, in the last few years, many large companies became interested in developing applications using agent technologies, and deployed applications for domains such as manufacturing, electronic commerce, process control, telecommunication systems, traffic and transportation management, information filtering and gathering, business process management, defense, entertainment and medical care [Luck et al., 2003].

The rest of this chapter is organized as follows. Section 2.1 describes the features and properties which characterize an agent. Section 2.2 presents an overview of a particular agent-oriented methodology – Tropos. Section 2.3 provides some insights into software design patterns, as well as the main design patterns proposed in agent-oriented community are reviewed. Finally, section 2.4 summarizes this chapter

2.1 Agent-Oriented Paradigm

Although many different perspectives of agency have been described and discussed, there is no universally accepted definition of what exactly determines an agent. In this work we address the agent definition as follows [Wooldridge, 2002]:

“An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives”

Based on our bibliographical research (cf. [Weiss, 2000] [Sturm and Shehory, 2003] [Green et al., 1997] [Muller, 1998] [Wooldridge and Jennings, 1995]), we have established the following agent properties as the ones that need to be considered in order to build an agent-oriented software system [Silva, C. et al., 2003] [Silva, C. et al., 2004]:

- *Autonomy*: ability of the software to act independently without direct intervention from humans or other agents. Active autonomous entities are not necessarily compliant with external demands or desires [Yu, 2002].
 - *Deliberativity*: A deliberative agent makes decisions by considering both information from environment and information about previous experiences. Such agents are able to act rationally in order to achieve goals.
 - *Reactivity*: agents perceive their environment, and respond in a timely fashion to changes that occur. It should be noted that reactive agents only act in response to external stimuli.
- *Organisation*: An organisation consists of a group of agents whose behaviour and relationships are regulated by a set of social norms intended to create a unit capable of achieving a common goal [Dignum and Dignum, 2001].
- *Sociability*: ability to participate in multiple relationships, interacting with a number of other agents, either simultaneously or at different times [Yu, 2002].
- *Interaction*: ability to communicate with the environment as well as with other agents. Agent’s interaction can be classified as:
 - *Communication*: ability to exchange messages with other agents;
 - *Cooperation*: ability to interact with other agents to achieve a common purpose; happens among non-antagonistic agents that either succeed or fail together;

- *Competition*: ability to interact with other agents where the success of one agent implies the failure of others (opposite of cooperation);
 - *Coordination*: ability to perform some activity in a shared environment with other agents, determining goals they share and common tasks, avoiding unnecessary conflicts and pooling resource [Weiss, 2000].
- *Negotiation*: ability to interact with other agents in order to reach an agreement about some matter. It involves the exchange of information, the relaxation of initial goals, mutual concessions, lies or threats [Green et al., 1997].

Although agent orientation has already been used in commercial and industrial applications [Luck et al., 2003], many multi-agent systems are still developed in an *ad hoc* manner while others uses methodologies under development, such as GAIA [Zambonelli et al., 2003], AUML [Odell et al., 2000], and Tropos [Castro et al., 2002] [Giorgini et al., 2005]. In fact, these methodologies may differ in their underlying premises, phases, models, concepts as well as in the way they deal with a set of properties present in most agent-oriented systems. A comparison between GAIA and Tropos with respect to these aspects has been provided in Silva, C. et al. (2004). In fact, since Tropos covers more phases than GAIA, in this work, we intend to improve the detailed design phase of Tropos methodology.

2.2 Tropos

The increasing interest in software agents and multi-agent systems has recently led to the development of new methodologies based on agent concepts. In particular, we are working on the improvement of Tropos [Castro et al., 2002] [Giorgini et al., 2005] - a requirements-driven framework aimed at reducing as much as possible the impedance mismatch between the system and its environment and building software that operates within a dynamic environment.

Tropos supports five phases of software development:

- Early requirements: early requirements analysis focuses on the intentions of stakeholders. It is concerned with the understanding of a problem by studying an organisational setting.

- Late requirements: the system-to-be is described within its operational environment, along with relevant functions and qualities (e.g. performance, security, availability). Late requirements analysis results in a requirements specification which describes all functional and non-functional requirements for the system-to-be.
- Architectural design: the system's global architecture is defined in terms of subsystems, interconnected through data, control and dependencies. A specific architectural style is to select among alternatives using as criteria the desired qualities identified in the Late Requirements phase.
- Detailed design: the detailed design phase is intended to introduce additional detail for each architectural component of a system. It consists of defining how the components present in the architectural model are going to fulfill its responsibilities according to design patterns.
- Implementation: the production of code from the detailed design specification, according to the established mapping between the implementation platform constructs and the detailed design abstractions.

2.2.1 Tropos Concepts and Models

Tropos adopts the concepts and models offered by the *i** [Yu, 1995] framework which includes concepts such as *actor* (actors can be *agents*, *positions* or *roles*), as well as social dependencies among actors including *goal*, *softgoal*, *task* and *resource* dependencies:

- Actor: an actor is an active entity that carries out actions to achieve goals by exercising its know-how.
- Agent: is an actor with concrete physical manifestations (a person or a system).
- Role: is an abstract characterization of the behaviour of a social actor within some specialized context, domain or endeavor.
- Position: is a set of roles typically played by one agent.
- Dependency: a dependency describes an intentional relationship between two actors, i.e., an “agreement” (called *dependum*) between two actors: the depender and the

dependee, where one actor (dependor) depends on another actor (dependee) on something (dependum).

- Dependor: the dependor is the depending actor.
- Dependee: the dependee is the actor who is depended upon.
- Dependum: the dependum is the type of the dependency and describes the nature of the agreement.
- Goal: a goal is a condition or state of affairs in the world that the stakeholders would like to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered.
- Softgoal: a softgoal is a condition or state of affairs in the world that the actor would like to achieve, but unlike in the concept of (hard) goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective judgment and interpretation of the developer to judge whether a particular state of affairs in fact achieves sufficiently the stated softgoal.
- Resource: a resource is an entity (physical or informational), with which the main concern is whether it is available.
- Task: a task specifies a particular way of doing something. Tasks can also be seen as the solutions in the target system, which will satisfy the softgoals (operationalizations). These solutions provide operations, processes, data representations, structuring, constraints and agents in the target system to meet the needs stated in the goals and softgoals.

As shown in Figure 2.1, actors are represented as circles; dependums -- goals, softgoals, tasks and resources -- are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form $\text{dependor} \Rightarrow \text{dependum} \Rightarrow \text{dependee}$.

The i* framework includes the Strategic Dependency (SD) model (Figure 2.1) for describing the network of relationships among actors, as well as the Strategic Rationale (SR) model (Figure 2.2) for describing and supporting the reasoning that each actor

goes through concerning its relationships with other actors by using a means-ends analysis.

The Strategic Dependency Model consists of a set of nodes and the links connecting them, where nodes represent actors and each link indicates a dependency between two actors. Hence, a model is described in terms of a network of dependency relationships among various actors, capturing the motivation and justification of activities. SD encourages a deeper understanding of a business process by focusing on intentional dependencies among actors, beyond the usual understanding based on activities and entity flows. It helps identifying what is at stake, for whom, and what impacts on the system are likely if a dependency fails. Although a strategic dependency model provides hints about why a process is structured in a certain way, it does not sufficiently support the process of suggesting, exploring, and evaluating alternative solutions. That is the role of the Strategic Rationale model.

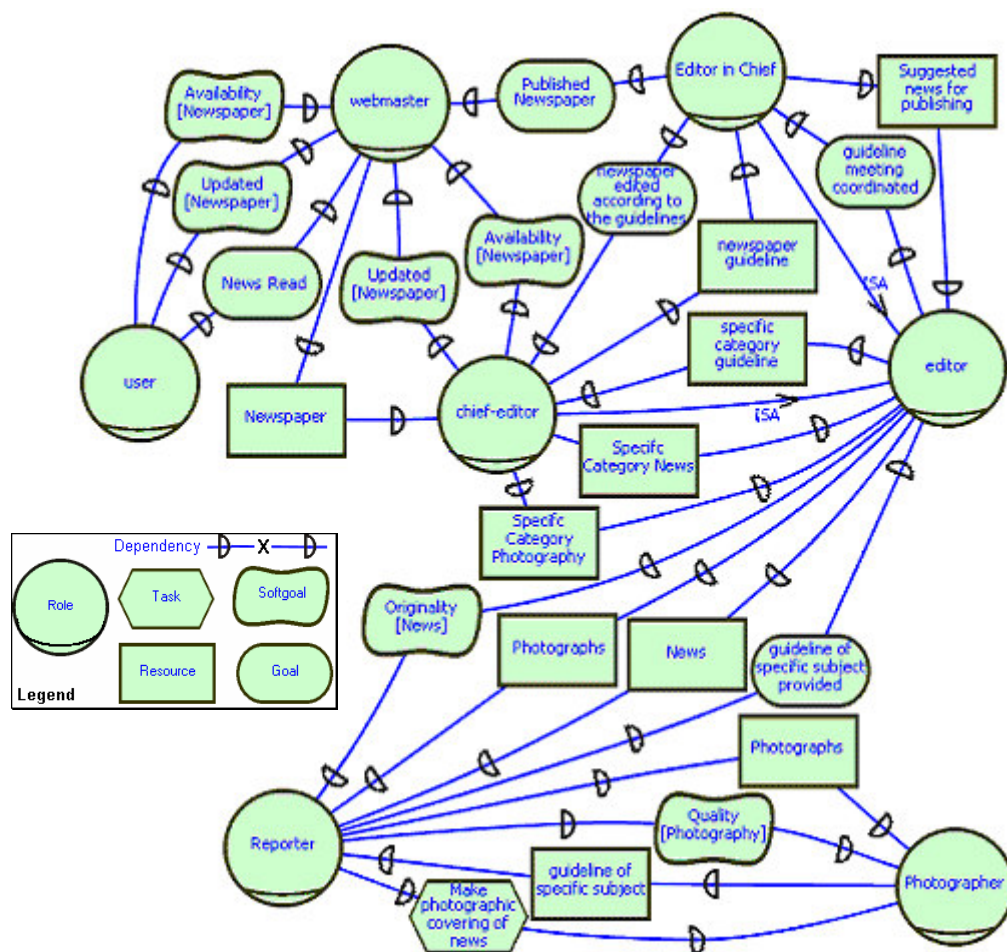


Figure 2.1 – Strategic Dependency Model for Newspaper Office

Figure 2.1 models the organizational environment of a Newspaper Office. When a user wishes to read news, he/she can access the newspaper website maintained by a Webmaster who is responsible for updating the published information. The newspaper to be published is edited and provided to the webmaster by the Chief Editor. The Chief Editor depends on each Editor to receive the news, which are then included in the newspaper, prepared according to the newspaper guideline. The newspaper guideline is generated by a meeting among the Editors, the Chief Editor and the Editor in Chief. In order to produce the newspaper it is necessary for the Chief Editor to share the newspaper guideline among the Editor agents, which are responsible for getting news of a specific category. For example, an Editor may be responsible for political news while another one may be responsible for sports news. Each Editor contacts one or many reporters which can find the news of specific subjects (e.g., basketball) to fulfill its sub-guideline (e.g., about sport news). Then, each reporter has to contact a photographer to make the photographic support of the news. The Chief Editor then edits, according to the guideline, the news provided by each Editor and forwards them to the Webmaster to publish them on the website.

The Strategic Rationale model (see Figure 2.2) is a supplementary model to the Strategic Dependency model by explicitly representing means-ends relationships. It provides a systematic way for exploring the space of possible new process designs. This model allows the modelling of the reasons associated with each actor and their dependencies. Each actor possesses external dependencies with other actors, but internally each one possesses goals and tasks that impel and justify each dependency. A good way to begin the decomposition is to observe how the *dependee* actor can satisfy the *dependum* associated with the same, and then to investigate and decompose intentions and strategic organisational reasons as a whole.

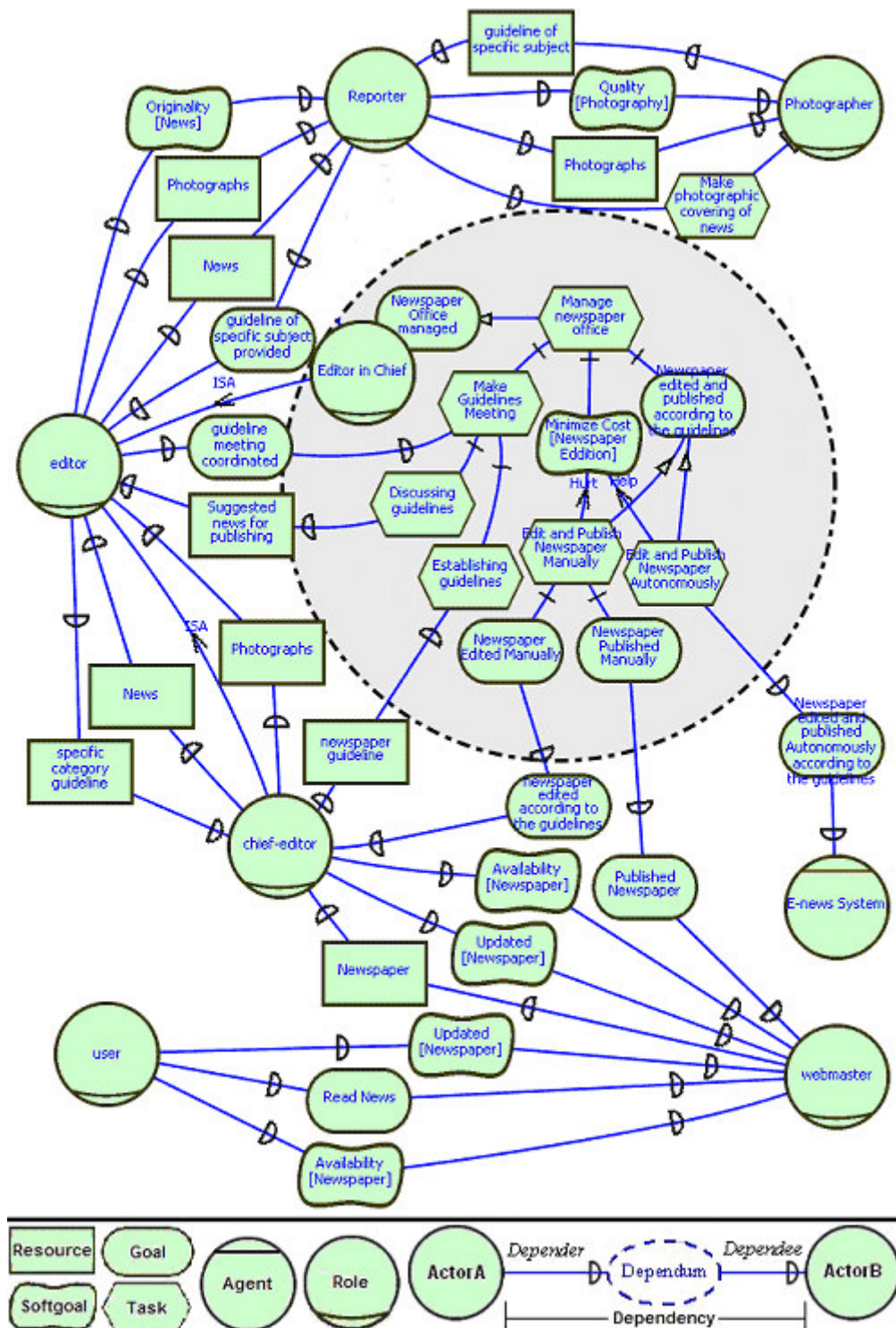


Figure 2.2 – Strategic Rationale model for Editor in Chief actor

A Strategic Rationale model is a graph with four main types of nodes -- goal, task, resource, and softgoal -- and two main types of links -- means-ends links and task decomposition links (see Figure 2.2). These links are described as:

- Means-ends link: this link is associated with obtaining a certain end, which can be a goal, resource, softgoal or task. The means to obtain the end is defined as tasks that are necessary to reach the end. In Figure 2.2, for example, the end element is the *Newspaper Office Managed* goal while the means element is the *Manage Newspaper Office* task. The contribution link is a special kind of means-end link. A contribution link from any element to a softgoal indicates a qualitative effect. This effect can be sufficient to satisfy/deny a softgoal (make/break), can have a positive/negative contribution which is in itself not sufficient to satisfy/deny a softgoal (help/hurt), can have a positive/negative contribution of unknown strength (some+/some-), or can have an unknown effect (unknown).
- Task-decomposition link: a node task is linked to its component elements through a decomposition connection. The four types of nodes can be linked. A task may be decomposed into smaller units of a node to represent the way and the reasons associated with the accomplishment of the task. For example, in Figure 2.2, the *Manage Newspaper Office* task is decomposed into *Make Guidelines Meeting* task, *Minimize Costs [Newspaper Edition]* softgoal and *Newspaper Edited and Published According to the Guidelines* goal.

Covering the complete agent-oriented software development lifecycle is considered as being essential to make agent orientation a mainstream paradigm. Among the various development phases supported by Tropos, we focus on the detailed design activity, which is presented briefly in the next section.

2.3 Agent-Oriented Design Patterns

In Tropos, the detailed design phase consists of defining how the goals assigned to each actor present in the architectural model, are fulfilled by agents performing roles according to design patterns.

Designers can be guided by a catalogue of multi-agent patterns which offer a set of standard solutions. Considerable work has been done in software engineering for defining software patterns (see e.g., [Gamma et al., 1994]). They focus on object-orientation not on the inherent intentional and social characteristics of agents. Accordingly, Tropos has defined a set of design patterns, named social patterns [Kolp et

al., 2002], focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems.

Design patterns are explicit formulations of good past software development experiences consisting of well-tested solutions to recurring problems that arise within some contexts and systems of forces. The patterns provide easy reuse of good software design. This concept is also becoming indispensable in developing practical large-scale agent-based software in a low cost by promoting reuse [Tahara et al., 1999]. Hence, agent-oriented community is also focusing its effort on applying design patterns to agent-based systems development. The objective is to minimise the cost of a project by capturing solutions to well-known (or common) problems in agent-oriented design.

2.3.1 Design Pattern Definition

In Alexander (1979), we find the following definitions of a pattern:

- A pattern is a three-part rule that expresses a relation between a certain context, a problem, and a solution.
- Each pattern is a relationship between a certain context, a certain system of forces that occurs repeatedly in that context, and a certain spatial configuration that allows these forces to resolve themselves.
- A pattern is an instruction, which shows how the spatial configuration can be used, repeatedly, to resolve the given system of forces wherever the context makes it relevant.
- The pattern is, in short, at the same time a thing, which happens in the world and the rule which tells us how to create that thing, and when we should create it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate the thing.
- A pattern is described in a set format to ease its dissemination. The format states the problem addressed by the pattern and the forces acting on it. There is also a context that must be present for the pattern to be valid, a statement of the solution, and any known uses [Kendall et al., 1998]. For example, Gamma et al. (1994) uses a template to describe object-oriented design patterns.

Although there are many formats for patterns, the minimal one contains the following, or essentially similar, headings [Deugo et al., 1999]:

- Name: a good pattern name should contain more information than just the number of words would suggest.
- Problem: a precise statement of the problem to be solved. A good problem for a pattern is one that software engineers will ask themselves often.
- Context: a description of a situation where the pattern can be applied to. Every pattern will have a number of forces that must be balanced before applying it. The context helps one determine the impact of the forces.
- Forces: a description of an item that influences or constrains the decision to apply the pattern in a context. A good pattern description should fully encapsulate all the forces that have an impact upon it.
- Solution: a description of the solution in the context that balances the forces.

Other sections such as rationale, resulting context, implementation, sample code, known uses and related patterns may be included to help with the description [Deugo et al., 1999].

In next section, we outline an overall description of the social patterns, i.e., a kind of design pattern that provides a social perspective on multi-agent systems [Kolp et al., 2005]. The interested reader can find a brief description of them in the literature.

2.3.2 Social Patterns

Kolp et al. (2002) proposed patterns focusing on social and intentional aspects that are recurrent in multi-agent and cooperative systems. The structures are inspired by the federated patterns introduced by Hayden and Yang (1999).

A multi-agent system can be seen as a *social organization* of autonomous software entities (agents) that can flexibly achieve agreed-upon *intentions* through their interactions. Taking real-world social behaviors as a metaphor, social patterns describe multi-agent systems as composed of autonomous agents that interact and coordinate their actions in order to achieve their intentions, just like actors in human organizations.

Moreover, social patterns are modeled according to five complementary dimensions: social, intentional, structural, communicational, and dynamic. AUML [Odell et al., 2000] is also used for modeling the last three dimensions.

The social patterns are classified into two categories [Kolp et al., 2005]: Pair and Mediation patterns. The Pair patterns describe direct interactions between negotiating agents and include:

- The **Booking Pattern** involves a client and a number of service providers. The client issues a request to book some resource from a service provider. The provider can accept the request, deny it, or propose to place the client on a waiting list, until the requested resource becomes available when some other client cancels a reservation.
- The **Subscription Pattern** involves a yellow-page agent and a number of service providers. The providers advertise their services by subscribing to the yellow pages. A provider that no longer wishes to be advertised can request to be unsubscribed.
- The **Call-For-Proposals Pattern** involves an initiator and a number of participants. The initiator issues a call for proposals for a service to all participants and then accepts proposals that offer the service for a specified cost. The initiator selects one participant to supply the service.
- The **Bidding Pattern** involves an initiator and a number of participants. The initiator organizes and leads the bidding process, and receives proposals. At every interaction, the initiator publishes the current bid; it can accept an order, raise the bid, or cancel the process.

The Mediation patterns feature intermediate agents that help other agents reach agreement about an exchange of services and include:

- In the **Monitor Pattern**, subscribers register for receiving, from a monitor agent, notifications of changes of state in some subjects of their interest. The monitor accepts subscriptions, request notifications from subjects of interest, receive such notifications of events and alerts subscribers to relevant events. The subject provides notifications of state changes as requested. The subscriber registers for notification

of state changes to distributed subjects, receive notifications with current state information, and update its local state information.

- In the **Broker Pattern**, the broker agent is an arbiter and intermediates the access to accessing services of an agent (provider) to satisfy the request of a consumer. The Broker agent locates providers corresponding to a Client request for service. It requests and receives the service from service providers and then forwards the service to the Client.
- In the **Matchmaker Pattern**, a matchmaker agent locates a provider corresponding to a consumer request for a service, and then hands the consumer a direct handle to the chosen provider. Contrary to the broker who directly handles all interactions between the consumer and the provider, the negotiation for service and actual service provision are two distinct phases.
- In the **Mediator Pattern**, a mediator agent mediates interactions among different agents. An initiator addresses the mediator instead of asking directly another colleague, the performer. The mediator has acquaintance models of colleagues and coordinates the cooperation between them. Conversely, each performer has an acquaintance model of the mediator. While a broker simply intermediates providers with consumers, a mediator encapsulates interactions and maintains models of initiators and performers behaviors over time.
- In the **Embassy Pattern**, an embassy routes a service requested by a foreign agent to a local one and handles the response back. If the access to the local agent is granted, the foreign agent can submit messages to the embassy for translation. The content is translated in accordance to a standard ontology. Translated messages are forwarded to target local agents. The results of the query are passed back out to the foreign agent, translated in reverse.
- The **Wrapper Pattern** incorporates a legacy system into a multi-agent system. The wrapper agent interfaces the clients to the legacy by acting as a translator between them. This ensures that communication protocols are respected and the legacy system remains decoupled from the rest of the agent system.

Chapter 4 presents the current approach to describe social pattern in Tropos. In particular, the matchmaker pattern will be used to illustrate it.

2.4 Summary

The characteristics of the agent-oriented paradigm are well suited to tackle the emerging complexities from software systems that are now required to operate in distributed, large, open, dynamic, unpredictable, heterogeneous and highly interactive application environments [Weiss, 2000].

Agent orientation is now being used in industrial and commercial applications (e.g. telecommunication, electronic and mobile commerce [Luck et al., 2003]). For a long time the agent-oriented development was ad-hoc and although many different methodologies have been proposed in the last few years, there is still no well-established development process to build agent-oriented applications. Hence, among the several concerns required for the consolidation of this new paradigm, we highlight the need for suitable techniques, notations and tools to support the agent-oriented software development.

Agent technology is still maturing and, therefore, there is no consensus about agent definition. This results in a set of different extensions of UML for the different agents features supported by the current implementation platforms [Kolp et al., 2005] [Odell et al., 2000] [Silva, V. et al., 2005]. One expects that as soon as agent paradigm becomes a widespread technology in the next years, a consensus in the industry and standardization agencies, such as Foundation for Intelligent Physical Agents (FIPA) standards [FIPA, 2004] and Agent Platform Special Interest Group [OMG, 2005a], determine what is the set of UML extensions which will be finally adopted for the detailed design of multi-agent systems.

There is much work still to be done in the area of software agent engineering, particularly in the area of defining design patterns and methods for applying them. The efficient use of design patterns has the potential for significant benefit to the architectural process and for successful implementation of multi-agent systems [Hayden and Yang, 1999]. In fact, developing systems using design patterns can contribute partially to the achievement of some quality requirements such as reusability, understandability, maintainability and evolvability. However, current software

development paradigms (such as objects and agents) can not lead properly to the scattering and tangling of design pattern concerns with the application functional modules concerns, failing the quality requirements aforementioned [Noda and Kishi, 2001]. Thus, it becomes clear that design patterns concerns can be considered as crosscutting concerns. To support advanced separation of design patterns concerns we can use abstractions and mechanisms provided by the aspect-oriented development [Kiczales et al., 1997], which provides support for separating the system functionality from its crosscutting concerns. The aspect-oriented paradigm is theme of the next chapter.

3 Aspect-Oriented Software Development

Crosscutting concerns are properties that do not align well with the decomposition criteria of the chosen approach and, therefore, cannot be modularized. These crosscutting concerns can be better addressed by adopting aspect-oriented software development techniques. The purpose of this new technology is to localize crosscutting concerns, avoiding their scattering and tangling through software artifacts. This chapter discusses the current state of the art for engineering aspect-oriented software systems.

Separation of concerns is a well-established principle in software engineering that addresses the limitations of human cognition for dealing with software complexity. A *concern* is some part of the problem that we want to treat as a single conceptual unit [Dijkstra, 1976]. The separation of concerns principle states that a given problem involves different kinds of concerns, which should be identified and separated to cope with complexity, and to achieve the required engineering quality factors such as robustness, adaptability and reusability [Aksit et al., 2001]. The basic idea is to handle one system concern separately. As a consequence, applying this principle allows to [Ghezzi et al., 1991]:

- decrease the software development complexity by concentrating separately in different concerns;
- reason about different concerns in a relatively independent way;
- divide and separating responsibilities between development team members; and
- improve modularity of software artifacts.

Moreover, when a complex software system has the separation of concerns property, it is divided into a set of highly cohesive modules (similarity of the module concerns) with a low coupling among them (minimal dependency among modules). Therefore, changing a module related to a specific concern tend to produce a limited effect – or even no effect – into modules related to other concerns [Hirsch and Lopes, 1995]. This improves the understandability, evolvability, adaptation, customization and reuse of software artifacts.

However, to incorporate the separation of concerns principle in both the development process and software artifacts, proper software engineering methods are required [Ghezzi et al., 1991]. These methods are usually based on the adopted programming paradigm. In agent oriented methods, for example, concerns are separated into roles and agents; in object oriented methods, the concerns are separated into objects and classes; in structured methods, the concerns are separated into modules and subsystems.

A good practice to represent concerns separately is to follow the “Law of Demeter” [Lieberherr and Holland, 1989], which determines that each unit should have only limited knowledge about other units, i.e., only those units “closely” related to the current unit. The mechanisms provided by existing software development paradigms (structured, object-oriented, agent-oriented) are not sufficient to modularize some

3.1 Aspect-Oriented Paradigm

The aspect oriented paradigm [AOSD, 2003] consists of a set of common concepts, mechanisms and practices for separation of crosscutting concerns. This paradigm has the purpose of localizing crosscutting concerns, avoiding their scattering through software artifacts or tangling these artifacts with specific concerns [Elrad et al., 2001].

The aspect-oriented paradigm represents an advance related to the previous paradigms, since it contributes to improve the separation of concerns in the software artifacts and, therefore, allows to achieve the benefits of applying the separation of concerns principle through the software development process.

The aspect oriented paradigm complements previous paradigms, such as structured, object-oriented and even so agent-oriented, in two senses [Elrad et al., 2001]: (i) localizing crosscutting concerns into separated units; and (ii) providing mechanisms to perform composition of crosscutting concerns with the units they cut across (called base units).

Figure 3.2 depicts the characteristics of aspect oriented paradigm: the crosscutting behavior is centralized into only one unit, known as aspect (Figure 3.2a); the composition mechanism is responsible for joining non-invasively the aspect behavior with the units it affects (Figure 3.2b), i.e., avoiding that the crosscutting concern to be explicitly scattered into the units it affects, as well as its tangling with the units' own concerns.

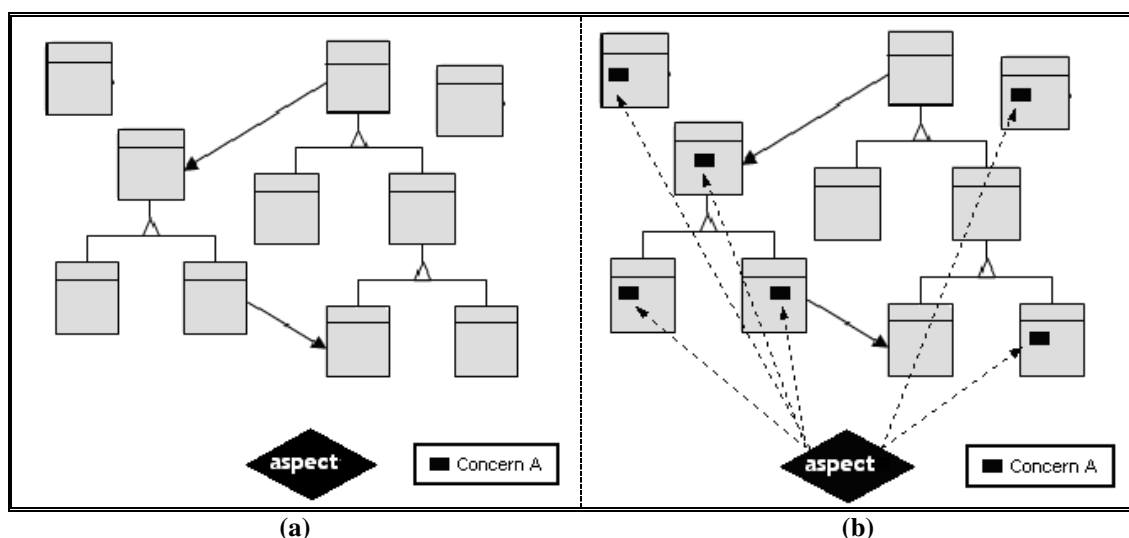


Figure 3.2 – Illustration of the separation and composition of aspects [Sousa, 2004]

To weave an aspect with the base units it affects, it is necessary to define a:

- Join point: indicating the point where the crosscutting concern should join the behaviour of the base unit(s) (a noncrosscutting concern such as a class) it cuts across [Kiczales et al., 2001];
- Composition rule: expressing the sequential order into which each aspect must be composed with the base unit(s) it cut across [Kiczales et al., 2001]. That is, it specifies how the aspect behaviour must be applied in the join point(s). The most common constructors of composition rules are: before, after and around the join point [Kiczales et al., 2001]:
 - *Before*: the composition happens whenever a join point is reached and before the actual computation proceeds;
 - *After*: the composition happens after the computation “under the join point” finishes, i.e. after the method body has run, and just before control is returned to the caller;
 - *Around*: the composition happens whenever a join point is reached, and has explicit control whether the computation under the join point is allowed to run at all.

Four different associations may arise between aspects and base units [Kersten and Murphy, 1999]:

- *closed* association: neither the aspect, nor the base unit, knows about the other.
- *open* association: both the aspect and the base unit know about each other.
- *Base unit-directional* association: the aspect knows about the base unit, but the base unit does not know about the aspect.
- *aspect-directional* association: the base unit knows about the aspect but the aspect does not know about the base unit.

The closed association is preferential, since it allows independency from both the base units and the aspect, promoting better potential to achieve reusability, understandability

and maintainability in software artifacts. The open association must be avoided, since it results in coupling between the two artifacts and, therefore, reducing the achievement of separation of concerns principle benefits. In its turn, the association and aspect-directional association, promote the qualities aforementioned only with relation with the base unit and the aspect, respectively. The most common type of associations in aspect oriented approaches are: closed and base unit-directional.

3.2 Aspect-Oriented Modeling in Software Design

The aspect-oriented software development (AOSD) has the purpose of handling crosscutting concerns explicitly, providing means to their systematic identification, modularization, representation and composition during all phases of software development lifecycle. In AOSD, crosscutting concerns are encapsulated into separated units, called aspects, in order to promote the localization of these concerns. It results in an improved modularity, which contributes to achieve software artifacts qualities such as reusability, maintainability and understandability and, consequently, the reduction of development costs.

Aspect- oriented modeling requires the use of a higher-level aspect model that addresses the aspect oriented programming concepts at a preliminary design stage (avoiding language specific details), and allows the designer to work at a more abstract level during software construction and evolution. It is expected that the design model conforms to design principles (low coupling, high cohesion, etc), facilitates the specification of systems in the presence of multiple aspects and crosscutting, promoting understandability and reducing the design-implementation gap in situations where aspect- based implementation languages are used [Chavez et al., 2005].

Next we will discuss some approaches that support aspect-orientation in modeling of software design.

3.2.1 ASideML

The aSideML [Chavez, 2004] is an aspect-oriented modeling language that provides notation, semantics and rules for specifying aspects and crosscutting at the design level. The aSideML language allows the designer to build models that focus on keys concepts, mechanisms and properties of aspect oriented systems, in which aspects and

crosscutting are explicitly treated as first-class citizens. Some of these models present a detailed design view of aspect oriented systems that may also serve as preliminary blueprints to be evolved towards the implementation models of aspect oriented programming languages and tools.

User-defined aSideML model elements can be structural or behavioral. The main structural model elements are aspects, crosscutting interfaces, crosscutting features, base elements (elements that aspects are supposed to enhance) and the relationships between them.

Aspects are defined as *parameterized* elements with one or more explicit *crosscutting interfaces* to organize join point descriptions and aspect behavior. Aspects abstract over the identity of the elements they will eventually crosscut, by declaring template parameters to hold actual names of classes and methods. A new kind of relationship, the *crosscutting relationship*, subsumes a relationship between an aspect and a base element; it also performs a binding that defines the base elements and operations that replace the aspect's template parameters. The detailed semantics of aSideML aspects are presented in Chavez (2004).

Figure 3.3 presents the design of the Observer pattern [Gamma et al., 1994] using the aSideML notation. Crosscutting features are listed in different compartments of crosscutting interfaces, depending on the kind of enhancement they support. The *Additions* compartment lists data and operations to be introduced in classes. The *Refinements* compartment lists crosscutting operations to be combined before, after or before/after class operations and the *Redefinitions* compartment lists crosscutting operations that override class operations. In these two compartments, each operation name (op) is adorned with the `_` symbol, with three permitted combinations: `_op`, `op_` and `_op_`. These adornments indicate that the crosscutting operation provides behavior to be combined before, after or before/after the base operation behavior. Finally, an optional compartment may be supplied to define placeholders for required operations (*Uses*). The aspect is presented with a small dashed rectangle superimposed on the upper right-hand corner of the rectangle for the aspect. This rectangle is a template parameter box that contains lists of formal parameters, one list for each aspect's crosscutting interface. The first parameter of each list is the name of the corresponding crosscutting interface.

The Observation aspect presented in Figure 3.3 has two crosscutting interfaces, one for each pattern participant. Observer is a crosscutting interface that modularizes crosscutting features that enhance arbitrary objects so that these elements become observers. The Subject crosscutting interface modularizes crosscutting features that enhance arbitrary objects so that these elements become subjects. The crosscutting relationships connect the Observation aspect to Button (binding stateChange to click) and ColorLabel (binding update to colorCycle).

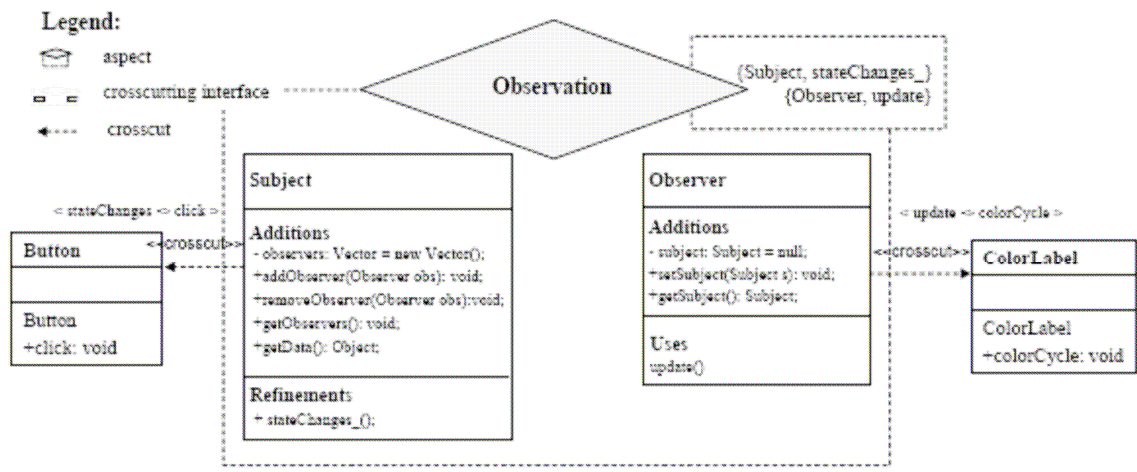


Figure 3.3 - Design-level aspects and crosscutting interfaces in aSideML.

Figure 3.4 illustrates the notation elements of the architectural model. The presentation of the architectural model is based on the example of the Observer design pattern. Architectural aspects are UML components [OMG, 2005b] represented as diamonds. Each interface is displayed as a small circle with the interface name placed next to the circle. The interfaces are attached to the architectural components, and are categorized in two groups: normal interfaces and crosscutting interfaces. Non-crosscutting interfaces are colored in white and crosscutting interfaces in gray.

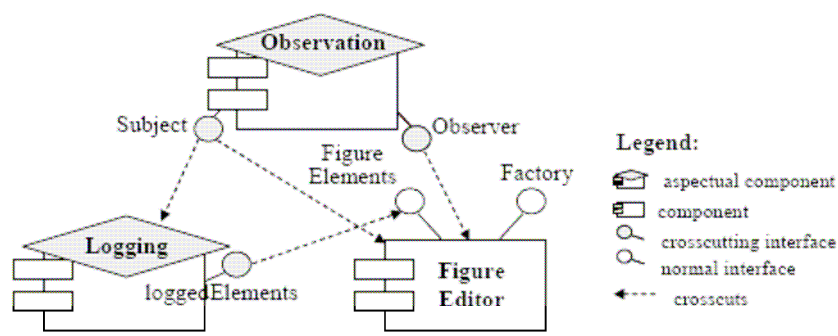


Figure 3.4 - Crosscutting Interfaces and Architecture-Level Aspects.

3.2.2 Pattern Specifications

Some approaches, such as Araújo et al. (2004), Georg et al. (2002), Whittle and Araújo (2004) and Clarke and Walker (2001), treat aspects as design patterns. In particular, Araújo et al. (2004) uses the Interaction Pattern Specification (IPSs) [France et al., 2004] to model aspectual scenarios.

Pattern Specifications (PSs) [France et al., 2004] are a way of formalizing the structural and behavioral features of a pattern. The notation for PSs is based on the Unified Modeling Language [OMG, 2005b]. A Pattern Specification describes a pattern of structure or behavior and is defined in terms of roles. A PS can be instantiated by assigning UML model elements to the roles in the PS. A pattern specification consists of a Structural Pattern Specification (SPS) that specifies the class diagram view of pattern solutions, and a set of Interaction Pattern Specifications (IPSs) that specifies interactions in pattern solutions. The SPS is the core of a pattern specification. The IPSs are expressed in terms of interaction participants specified in the SPS. A UML model conforms to a pattern specification if its class diagram conforms to the SPS and the interactions described by sequence diagrams conform to the IPSs.

An SPS defines the part of the pattern metamodel that characterizes class diagram views of pattern solutions. It consists of a structure of pattern roles [Kim et al., 2003] (henceforth referred to as roles), where a role specifies properties that a UML model element must have if it is to be part of a pattern solution model.

A role in an SPS can be classified as a classifier or a relationship role. A model role has a base metaclass in the Unified Modeling Language metamodel [OMG, 2005b]. For example, a model role whose base is Class metaclass is called a class role. The class role is played by classes (i.e., instances of Class) that satisfy the constraints specified in the role. A classifier role has structural and behavioral properties which are also roles whose bases are UML StructuralFeature and BehavioralFeature, respectively. The symbol “|” is used to indicate model roles. A UML model element conforms to (or plays) a role if it satisfies the properties defined in the role. The bindings are indicated by the dashed lines between the class diagram and the SPS in Figure 3.5 (e.g., Kiln is bound to the Subject role). The class Kiln describes kiln objects whose temperatures are monitored by TempObs objects.

An Interaction Pattern Specification defines a pattern of interactions between its participants. It consists of a number of lifeline roles and message roles, where each lifeline role is associated with a classifier role. Figure 3.6 shows an example of an IPS and a conforming sequence diagram.

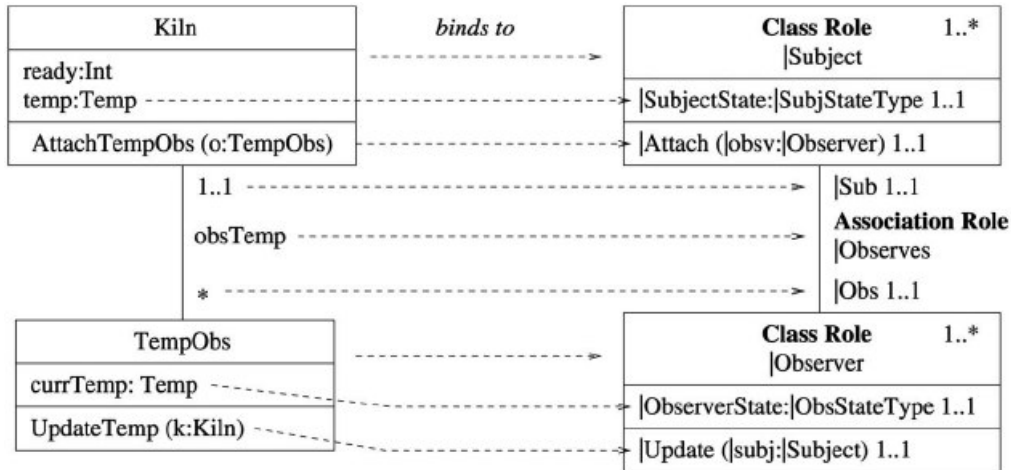


Figure 3.5 - An SPS and a structurally conforming class diagram. (a) A conforming class diagram and (b) the SPS [France et al., 2004]

The IPS in Figure 3.6(a) formalizes the Observer pattern. Role names are preceded by a vertical bar to denote that they are roles. An IPS can be instantiated by assigning concrete modeling elements to the roles. A conforming sequence diagram (see Figure 3.6(b)) must instantiate each of the roles with UML model elements satisfying the partial ordering on the message roles, multiplicity and other constraints (e.g., given in the Object Constraint Language (OCL) [OMG, 1997]).

In Figure 3.6, (b) conforms to (a) if we define instantiations as follows:

1. Bind `|NotifyInteraction` to `KilnInteraction`
2. Bind `|s` to `s`
3. Bind `|Subject` to `Kiln`
4. Bind `|o[i]` to `t[i]`
5. Bind `|Observer` to `TempObs`
6. Bind `|Notify` to `NotifyObs`

7. Bind *|Update* to *UpdateTemp*
8. Bind *|GetState* to *GetKilnTemp*
9. Bind *lst* to *st*
10. Bind *NumOfObservers* to *NumOfTempObs*

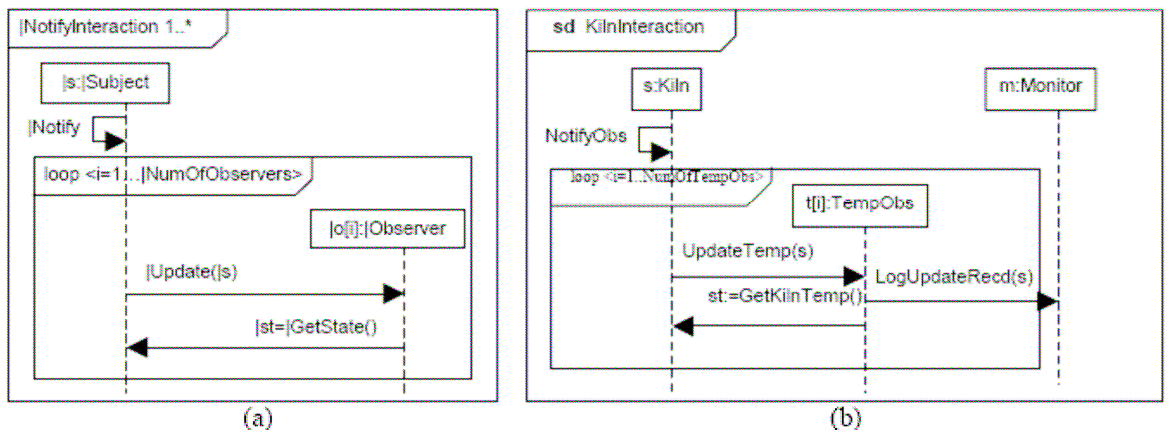


Figure 3.6 - An IPS (a) and a Conforming Sequence Diagram (b) [France et al., 2004]

The approach presented in Araújo et al. (2004), represent aspectual scenarios as IPS. In [France et al., 2004] an IPS consists solely of role elements. However, Araújo et al. (2004) extends this definition to allow an IPS to contain both role elements and non-role (i.e. concrete modeling) elements. An example of this in Figure 3.6(a) would be if the *|Subject* role were replaced with the concrete modeling element, *Kiln*, in the IPS. Allowing non-role elements in an IPS gives much greater flexibility in specifying aspects. The inclusion of concrete modeling elements in the pattern specifications reduces the number of instantiation steps since roles that would be instantiated to the same elements in all contexts can be directly represented as those elements in the pattern specification.

3.2.3 Multi-agent Systems Design

Garcia (2004) argues that the internal architecture of a single agent encompasses multiple concerns, including interaction, adaptation, autonomy, knowledge, collaboration, roles, learning, and mobility. As a consequence, multi-agent systems engineers are concerned with issues such as making an agent interact appropriately, handling the agent's adaptive behavior, structuring the agent's autonomous behavior,

and incorporating learning mechanisms into the agent's structure. Each property interacts with an agent's basic functionality and often with other agent properties. It is difficult to promote separation of agent concerns based on only object oriented abstractions and methods. The design and implementation of agent concerns tend to affect or crosscut many classes of the system design and code, including those representing the basic agent functionality. The lack of modularization introduces undesirable problems, such as higher coupling and lower cohesion in the system components. To address this issue, Garcia (2004) proposes an aspect-oriented approach to support the separate handling and modularization of multi-agent systems specific concerns. *Aspects* were used as unifying abstractions to capture the agent concerns that are hard to modularize with existing architectural abstractions and object-oriented software engineering.

The proposed approach encourages the separate handling of agent properties, and provides a disciplined scheme for their composition. The approach is composed of an architectural method, a pattern language, and an assessment framework. The architectural method and the pattern language provide aspect-oriented solutions for modularizing the agent concerns at different stages of design and implementation. The purpose of the assessment framework is to support the evaluation of the reusability and maintainability of aspect-oriented solutions based on a metrics suite and a quality model. Experimental studies in different application domains have been conducted to assess the proposed approach based on qualitative and quantitative criteria.

Figures 3.7 and 3.8 show the modeling of a multi-agent system, named Expert Committee (EC), using the aSideML notation. EC is an open multi-agent system that supports the management of paper submissions and the reviewing process for a conference. Software agents have been introduced to the EC system in order to assist researchers with timeconsuming activities in the paper submission and reviewing processes. EC agents are software assistants that represent paper authors, chairs, PC members and reviewers and coordinate their activities.

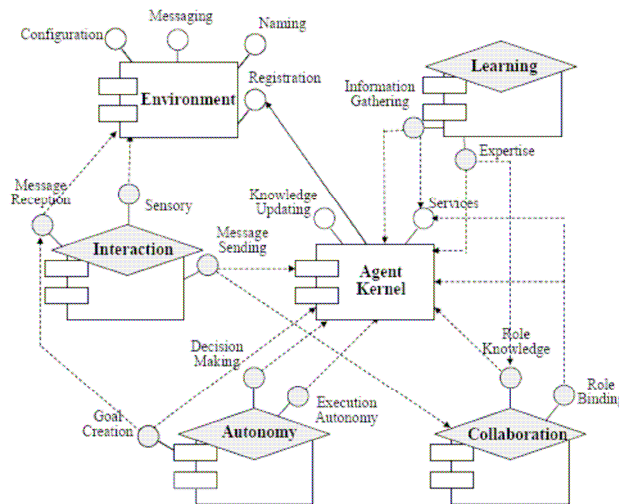


Figure 3.7 - The Aspect-Oriented Architecture of EC Agents [Chavez et al., 2005]

The architectural model for the software agents in the EC system is depicted in Figure 3.7, and then the detailed design of some architectural components is depicted in Figure 3.8. Figure 3.7 introduces the model of the AO agent architecture that encompasses the components for aspectizing common crosscutting concerns in multi-agent systems, such as learning and collaboration. The AO architecture is composed of two main normal components. The Environment component represents the agent location and the system services, such as naming service, registration, communication, and so forth. The Kernel component encapsulates the basic services provided by the agent for its clients; these services are non-crosscutting. As a result, this component realizes the Services normal interface to make those basic agent functions available to the external entities. This component is also responsible for modularizing the knowledge elements, such as actions, plans, goals, and beliefs. The KnowledgeUpdating interface is used to alter and evolve the internal agent knowledge [Chavez et al., 2005].

There are also aspectual components that separate the crosscutting agent-related concerns from each other and from the Kernel component. Most of the aspectual components crosscut multiple agent components in different ways, capturing their crosscutting property. As a result, they realize more than one crosscutting interface and also affect other architectural aspects. For example, the MessageSending interface crosscuts the Kernel component and the Collaboration architectural aspects [Chavez et al., 2005].

Figure 3.8 illustrates only the detailed design of the Kernel, Learning and Collaboration components. The Kernel component is refined as a set of classes, which represent the

agent itself, and knowledge elements (e.g. plans). Learning aspects are heterogeneous aspects that encapsulate the entire implementation of the learning concern. Their heterogeneity is mastered by the Expertise interface and the InformationGathering interface. These crosscutting interfaces are defined by the Learning abstract aspect and specialized by the ReviewerLearning aspect (Figure 3.8) [Chavez et al., 2005].

The Collaboration component aggregates collaboration protocols and roles played by the agents during their collaborative activities. Each role is represented by a design aspect and, as a consequence, the Collaboration component is realized by a set of inner role aspects. It is composed by four inner aspects, each one for a specific agent role: Author, Reviewer, PCMember, and Chair. Figure 3.8 illustrates the Reviewer aspect. Each inner aspect implements the RoleBinding interface and the RoleKnowledge interface. The first interface determines the events wherein a given role is bound to the agent. The second interface defines a set of additions which comprises the role-specific knowledge introduced to the agent playing that role [Chavez et al., 2005].

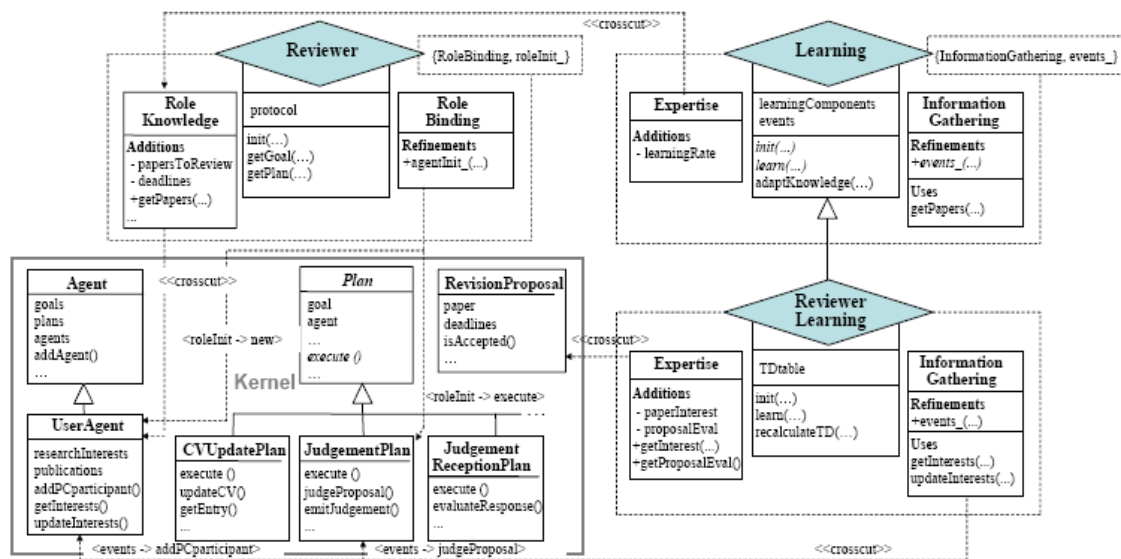


Figure 3.8 - Refining the Architectural Aspects and Crosscutting Interfaces
[Chavez et al., 2005]

3.3 Summary

Aspect orientation is an evolution of current software development technologies. Its purpose is complementing current paradigms in order to improve the separation of crosscutting concerns in software development, minimizing coupling between artifacts

and, therefore, providing better understandability, maintainability and reusability to the artifacts generated during software development lifecycle.

The aspect oriented paradigm is recent and, therefore, there is no consensus about the concepts and practices it encloses. Moreover, currently, a great number of proposals for aspect oriented software development is limited, i.e., refers to a specific phase of software development lifecycle. In particular, this thesis focuses on detailed design approaches for developing complex software systems. In this context, we have presented some modeling languages to capture aspect properties during software design [Stein, 2002] [Chavez, 2004] and for representing aspects as patterns [Araújo et al., 2004] [Whittle and Araújo, 2004] [France et al., 2004]. These latter approaches will be used as foundation of our technique to describe social patterns using advanced separation of concerns.

A particular approach, proposed by Garcia (2004), promotes separation of agent concerns (interaction, adaptation, autonomy, knowledge, collaboration, roles, learning, and mobility) based on aspect oriented abstractions and methods. In fact, aspects were used to capture the agent concerns that are hard to modularize with existing abstractions of object-oriented software engineering. This proposal uses the aSideML notation [Chavez, 2004].

We are also interested in multi-agent systems development. In particular, we are focusing on the MAS design using agent-oriented design patterns, called social patterns [Kolp et al., 2005]. In order to improve the social patterns description, we propose to use aspects as abstractions to capture the social patterns concerns that crosscut software agents composing the multi-agent system. Especially, the social patterns concerns are treated as aspects.

4 Social Patterns in Tropos

In this chapter we present the current state of art on social patterns. In particular, the lack of a detailed description of social patterns is discussed. Moreover, we illustrate the crosscutting nature of a specific social pattern and point out the limitations of the current detailed design phase of Tropos.

Software patterns have proved to be highly useful within the object-oriented field, and can make the applications more flexible, understandable, and reusable [Aridor and Lange, 1998]. Indeed, the efficient utilization of design patterns has the potential for bringing significant benefits to the architectural process and to the successful implementation of multi-agent systems [Hayden and Yang, 1999].

Tropos has proposed a set of design patterns focusing on social and intentional aspects [Kolp et al., 2002]. The so called *Social Patterns* offer a microscopic view of the multi-agent system at the detailed design phase to express in richer details each component present in the architectural design of the system [Kolp et al., 2005]. However, Tropos does not define a detailed description of social patterns, as well as a systematic way to choose and apply them in order to refine the architectural components of the software under development.

This chapter presents some limitations on related work which motivated this thesis. It is organized as follows. Section 4.1 describes the proposal of Kolp et al. (2005) to describe social patterns. Section 4.2 presents the crosscutting nature of a specific social pattern - the Matchmaker pattern (previously explained in section 2.3.2). Section 4.3 shows the current definition of Tropos detailed design phase. Finally, section 4.4 summarizes this chapter.

4.1 Social Patterns Description

In Kolp et al. (2005) a framework to explore the social patterns is proposed. It includes five different complementary dimensions to specify the MAS detailed design and to describe the social patterns, as follows:

- the *social dimension* identifies the relevant agents in the system and their intentional interdependencies.
- the *intentional dimension* identifies and formalizes services provided by agents to realize the intentions identified by the social dimension, independently of the plans that implement those services. This dimension answers the question: "What does each service do?"

- the *structural dimension* operationalizes the services identified by the intentional dimension in terms of agent-oriented concepts like beliefs, events, plans, and their relationships. This dimension answers the question: "How is each service operationalized?"
- the *communicational dimension* models the temporal exchange of events between agents.
- the *dynamic dimension* models the synchronization mechanisms between events and plans.

The social dimension specifies a number of agents and their intentional interdependencies using the i* model [Yu, 1995]. Figure 4.1 shows a social diagram for the Matchmaker pattern.

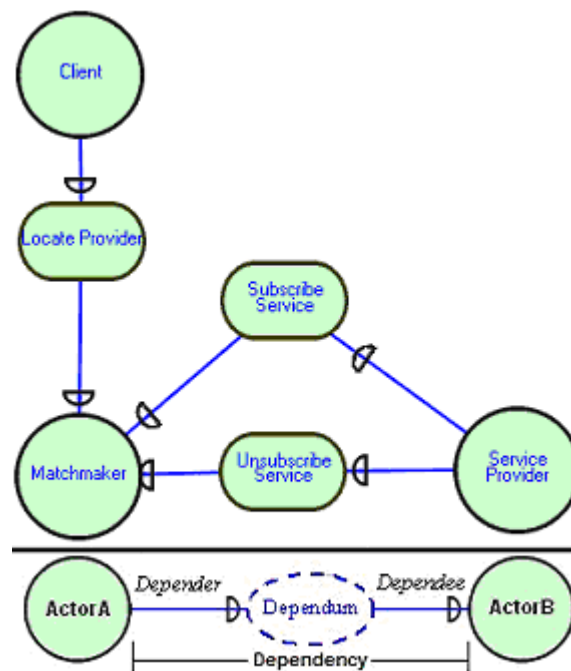


Figure 4.1 – Social Diagram for the Matchmaker Pattern [Kolp et al., 2005]

While the social dimension focuses on interdependencies between agents, the intentional dimension (Table 4.1) aims at modeling agent rationale. It is concerned with the identification of *services* provided by agents and made available to achieve the intentions identified in the social dimension. Each service belongs to one agent. Service definitions can be formalized by its fulfillment condition.

While the intentional dimension answers the question "What does each service do?", the structural dimension answers the question "How is each service operationalized?". Services are operationalized as *plans*, that is, sequences of actions. The knowledge that an agent has (about itself or its environment) is stored in its *beliefs*. An agent can act in response to the *events* that it handles through its plans. A plan is used by the agent to read or modify its beliefs, and send events to other agents or post events to itself. The structural dimension is modeled using a UML style class diagram extended for MAS engineering [Kolp et al., 2005].

Table 4.1 - Some services of the Matchmaker pattern

Service Name	Informal Definition	Agent
SendProviderRequest	Send a service request to a Matchmaker	Client
SendServiceRequest	Send a service request to a Provider	Client
SendProviderRequestResult	Send an answer to the Client	Matchmaker
SendServiceRequestResult	Send an answer to the Client	Provider
Send(un)SubscribeRequest	Send (un)subscription request to a Matchmaker	Provider
Send(un)SubscribeResult	Send an answer to the Provider	Matchmaker
LocateProviderIdentification	Search for some provider for a specific service	Matchmaker
PerformRequestedService	Perform the service requested for some Client	Provider

Figure 4.2 depicts the concepts and their relationships needed to build the structural dimension. Each concept defines a common template for classes of concrete MAS.

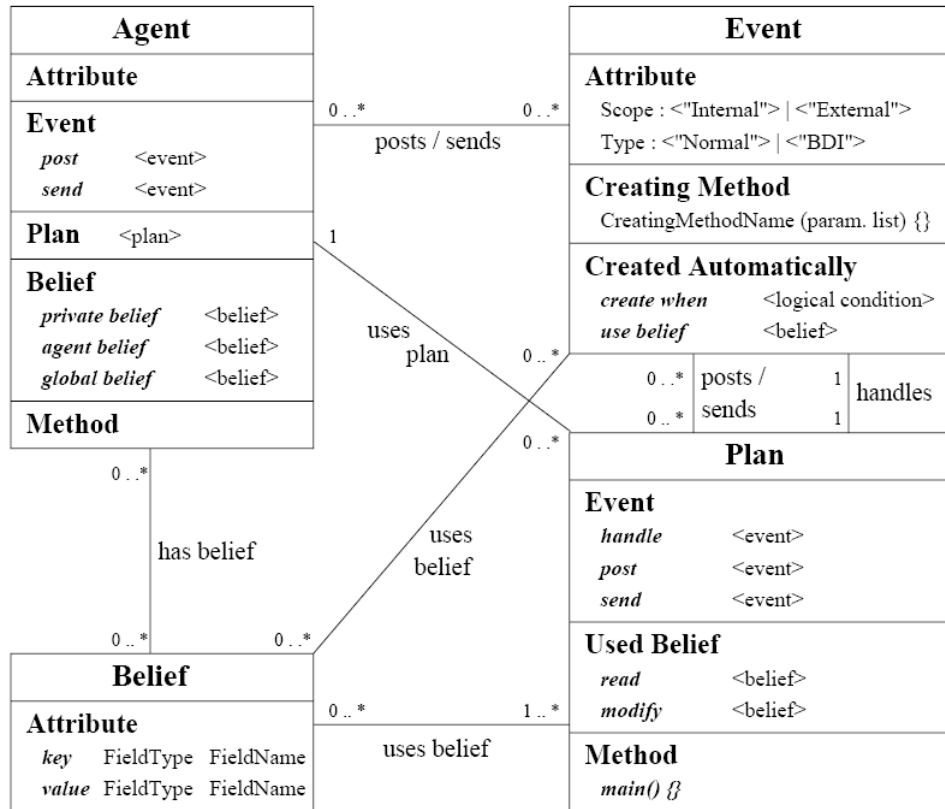


Figure 4.2 – Structural Diagram Template [Kolp et al., 2005]

According to Kolp et al. (2005) a **Belief** describes a piece of the knowledge that an agent has about itself and its environment. Beliefs are represented as tuples composed of a key and value fields. **Events** describe stimuli, sent by agents or automatically generated, in response to that the agents must take an action. Agents communicate with each other through events. As shown in Figure 4.2, the structure of an event is composed of three parts: declaration of the attributes of the event, declaration of the methods to create the event, declaration of the beliefs and the condition used for an automatic event. The third part only appears for automatic events. Events can be described along three dimensions:

- *External or internal* event: external events are sent to other agents while internal events are posted by an agent to itself. This property is captured by the *scope* attribute.
- *Normal or BDI (Belief-Desire-Intention)* event: an agent has a number of alternative plans to respond to a BDI event and only one plan in response to a normal event. Whenever an event occurs, the agent initiates a plan to handle it. If the plan execution fails and if the event is a normal event, then the event is said to

have failed. If the event is a BDI event, a set of plans can be selected for execution and these are attempted in turn. If all selected plans fail, the event is also said to have failed. The event type is captured by the *type* attribute.

- *Automatic or nonautomatic* event: an automatic event is automatically created when certain belief states arise. The *create when* statement specifies the logical condition which must arise for the event to be automatically created. The states of the beliefs that are defined by *use belief* are monitored to determine when to automatically create events.

A **Plan** describes a sequence of actions that an agent can take when an event occurs. A plan can be invoked by an event that handles and can create new events. As shown in Figure 4.2, plans are structured in three parts: the Event part, the Belief part, and the Method part. The Event part declares events that the plan handles (i.e., events that trigger the execution of the plan) and events that the plan produces. The latter can be either posted (i.e., sent by an agent only to itself) or sent (i.e., sent to other agents). The Belief part declares beliefs that the plan reads and those that it modifies. The Method part describes the plan itself, that is, the actions performed when the plan is executed [Kolp et al., 2005].

The **Agent** concept defines the behavior of an agent, as composed of five parts: the declaration of its attributes, of the events that it can post or send explicitly (i.e., without using its plans), of the plans that it uses to respond to events, of the beliefs that make up its knowledge, and of its methods.

The beliefs of an agent can be of type *private*, *agent*, or *global*. A *private* access is restricted to the agent to which the belief belongs. *Agent* access is shared with other agents of the same class, while *global* access is unrestricted.

Agents interact with each other by exchanging events. The communicational dimension models, in a temporal manner, events exchanged in the system. They adopt the sequence diagram model proposed in AUML [Odell et al., 1999] and extend it: *agent_name/role:pattern_name* expresses the role (*role*) of the agent (*agent_name*) in the pattern; the arrows are labeled with the name of the exchanged events.

Figure 4.3 shows a scenario for the Matchmaker pattern represented by a UML sequence diagram [OMG, 2005b]. The client sends a service request (Request ServiceProvider (SP) location) containing the characteristics of the service it wishes to obtain from a service provider. The Matchmaker may alternatively answer with a refusal (refuse request) or an acceptance (send SP location). The ServiceProvider sends a subscription in the Matchmaker's Yellow Pages. The Matchmaker may alternatively answer with a refusal (refuse subscription) or an acceptance (accept subscription). The ServiceProvider sends an unsubscription in the Matchmaker's Yellow Pages. The Matchmaker may alternatively answer with a refusal (refuse unsubscription) or an acceptance (accept unsubscription).

According to Kolp et al. (2005), a plan can be invoked by an event that handles and it can create new events. Thus, relationships between plans and events can rapidly become complex. To cope with this problem, they propose to model the synchronization and the relationships between plans and events with activity diagrams extended for agent-oriented systems (Figure 4.4). These diagrams specify the events that are created in parallel, the conditions under which events are created, which plans handle which events, and so on. An internal event is represented by a dashed arrow and an external event by a solid arrow. A BDI event may be handled by alternative plans. They are enclosed in a roundcorner box. Synchronization and branching are represented as usual.

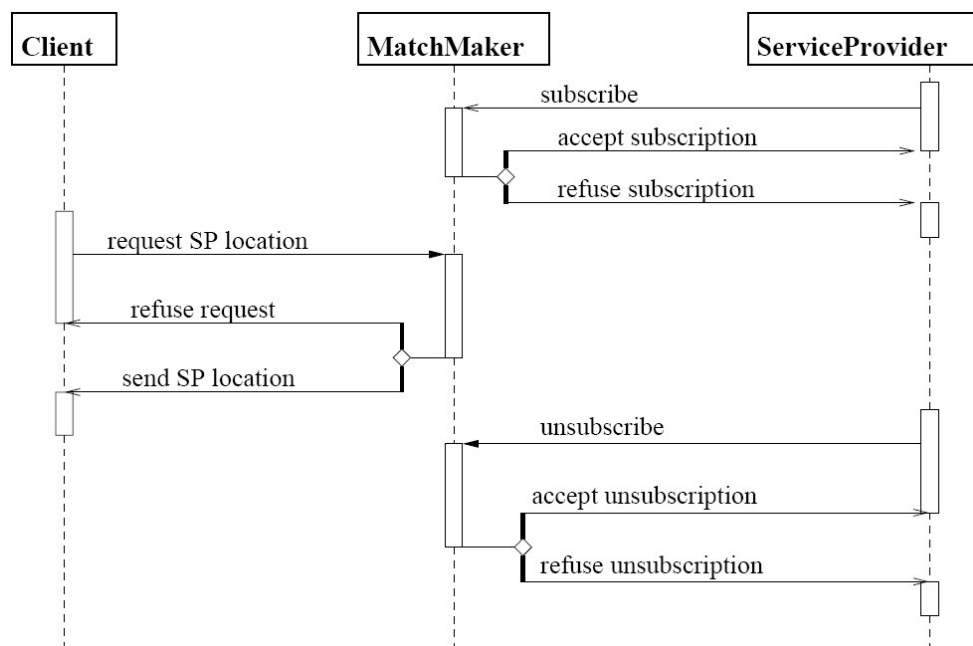


Figure 4.3 – Communication diagram for the Matchmaker pattern [Kolp et al., 2005]

For example, in Figure 4.4 the Client performs the `SendProviderRequest` plan. The Matchmaker receives the request and then performs the `LocateProviderIdentification` plan. If a provider has been identified, then the Matchmaker sends the provider identification to the Client. Otherwise, the Matchmaker sends the refusal for the Client request. If the Provider wishes to subscribe his services in the Matchmaker's Yellow Pages, the he sends a resquest by performing the `send(un)Subscription` plan. When the Matchmaker receives the request, he performs the `(un)SubscribeYellowPages` plan. If the service requested by the Client to the Provider is successfully executed, the Provider performs the `RecordAndSendSPInformDone` plan. In this case, the Client performs the `RecordPerformedService` plan. Otherwise, the Provider performs the `RecordAndSendSPFailure` plan. In this case, the Client performs the `RecordProviderFailure` plan.

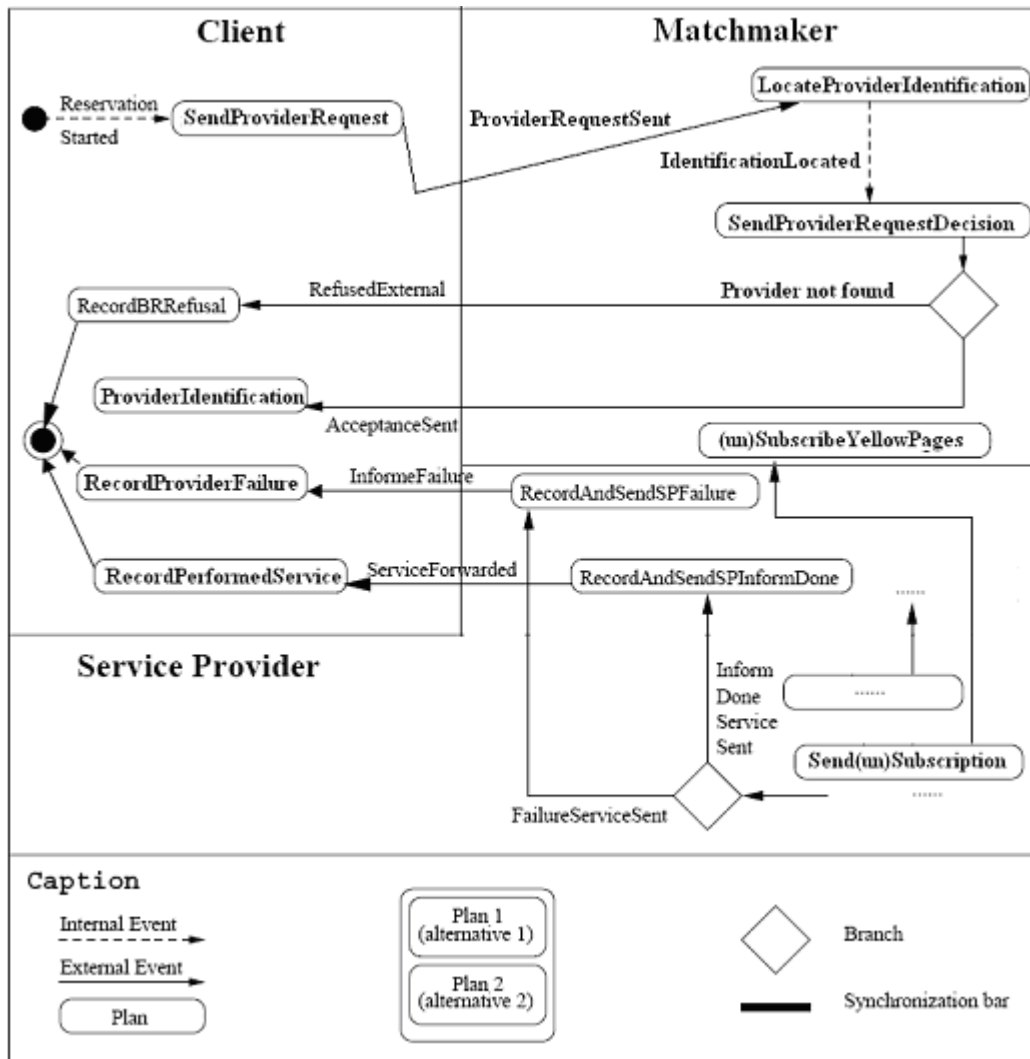


Figure 4.4 – Dynamic Diagram – Matchmaker

For agent-oriented programming, Kolp et al. (2005) proposes a code generator to automate the use of social patterns. It has been developed in Java and produces code for JACK [AOS, 2005], an agent-oriented development environment built on top of Java. JACK extends Java with specific capabilities to implement agent behaviors. In this code generator, the programmer first chooses which social pattern to use, then the roles for each agent in the selected pattern. The process is repeated until all relevant patterns have been identified. The code generator then produces the generic code for the patterns (.agent, .event, .plan, .bel JACK files).

Observe that the structural diagram is strongly dependent on the concepts supported by JACK, making easier the code generation. However, the detailed design of a system should be platform independent in order to increase the reuse in other software projects. Moreover that approach does not capture some MAS features, such as agent roles, norms, ontology, intention, rights to access resources in the environment, among others.

4.2 The Crosscutting Nature of the Matchmaker Pattern

Several design patterns for multi-agent systems have been proposed [Tahara et al., 1999] [Aridor and Lange, 1998] [Hayden and Yang, 1999] to describe recurring problems and solutions in software design and, therefore, improving software reusability and maintainability. However, in general, the use of design patterns by current software development paradigms (e.g., agent orientation), makes application core dependent on patterns, decreasing possibilities of reuse [Noda and Kishi, 2001]. In this section we show the crosscutting nature of the Matchmaker pattern by considering the implementation of each pattern participant concern (i.e., Client, Matchmaker and Provider concerns) provided by the JADE's API (Application Programming Interface) [Bellifemine et al., 2003]. JADE is a suitable agent platform to support the implementation of MAS. In JADE, a behavior represents a task that an agent can carry out.

One of the most important features that JADE agents provides is the ability to communicate and the DF (Directory Facilitator) agent, which implements the behavior of the main participant of the Matchmaker pattern [Kolp et al., 2005] i.e., the Matchmaker agent. It provides a Yellow Pages service by means of which an agent (Client) can find other agents providing the services it requires in order to achieve its

goals. An agent wishing to publish one or more services (Provider) must provide the DF with a description including its Agent Id (AID) and the list of its services. The services provided by the DF agent (Matchmaker) are usually used by all agents of a MAS implemented in JADE. Thus, all agents who need to register or unregister its services in the Yellow Pages of the DF agent will implement the Provider's concern (Figure 4.5 and Figure 4.6) [Silva, C. et al., 2006e].

```

DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(genericAgent.getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType(genericAgent.getServiceType());
sd.setName(genericAgent.getServiceName());
dfd.addServices(sd);
try {
    DFService.register(genericAgent, dfd);
}

```

Figure 4.5 – Code to register the service in the yellow pages

```

try {
    DFService.deregister(genericAgent);
}

```

Figure 4.6 – Code to unregister the service in the yellow pages

Similarly, all agents who need to find a specific provider agent in the Yellow Pages will implement the Client's concern (Figure 4.7).

Generally, the Yellow Pages Provider and Yellow Pages Client concerns (henceforth YPPProvider and YPClient concerns, respectively) are going to be implemented by several agents involved in the Matchmaker pattern. Consequently, the YPPProvider and YPClient concerns become tangled with the concerns of these agents (i.e., application core functionality) and scattered in the system. To solve this issue, we can use aspect-oriented abstractions both in the design and implementation phases.

```

DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(genericAgent.getServiceType());
template.addServices(sd);
try {
    DFAgentDescription[] result =
    DFService.search(genericAgent, template);
    for (int i = 0; i < result.length; ++i) {
        genericAgent.getProviders()[i] = result[i].getName();
    }
}

```

Figure 4.7 – Code to search in the yellow pages for the agent which provides a service

Observe that the concern of the Matchmaker participant, i.e. the *Yellow Pages service* concern, is implemented inside the DF agent and its implementation is hidden in the JADE's API. Thus, the Matchmaker participant concern will not crosscut the application functional modules.

In this context, we need a notation which supports the separation of crosscutting concerns in multi-agent system. This notation is required to both describe the social patterns and specify MAS detailed design. Although many languages for multi-agent systems modeling have been proposed in the last few years, they do not consider crosscutting concerns, such as AUMML [Odell et al., 2000], AORMML [Wagner, 2003], MAS-ML [Silva and Lucena, 2004] and even the proposal of Kolp et al. (2005). On the other hand, some languages for modeling separation of crosscutting concerns in object-oriented systems have been proposed, such as aSideML [Chavez, 2004], Composite Patterns [Clarke and Walker, 2001] and AODM [Stein, 2002], but they don't support agent constructs. In particular, Garcia (2004) uses the aSideML notation to support the separate handling and modularization of MAS specific concerns (e.g. interaction, adaptation, autonomy, knowledge, collaboration, roles, learning and mobility). However, his approach proposes developing MAS using object orientation and aspect orientation to design and implement the MAS specific concerns.

4.3 The Detailed Design Phase in Tropos

The detailed design phase is intended to introduce additional detail for each architectural component of a system. It consists of defining how the goals assigned to each actor present in the architectural model, are fulfilled by agent roles according to social patterns. In Wautelet et al. (2005) the detailed design process uses the approach of Kolp et al. (2005) to specify MAS and presents six activities:

- **Select a Social Pattern activity:** this aims at selecting a design pattern (if needed) for a project to solve a problem commonly found in software designs and prescribe a flexible solution for the problem.
- **Include New Goals activity:** this introduces new goals in the Social Dimension of MAS design based on the selected design pattern.
- **Identify Services activity:** this provides the Intentional Dimension by identifying the services provided by each agent that he can use to achieve the goal dependencies. Each service belongs to an agent and has to be specified during this activity.

- Specify Agent Structure activity: this provides the Structural Dimension where the structure of each agent and the agent-oriented concepts as Plans, Events and Beliefs are specified resulting in an UML class diagram extended with agent concepts.
- Represent Agents Communications activity: this aims at modeling, in a temporal manner, events exchanged in the system by using an extended AUMML sequence diagrams [Odell et al., 2005]. The result of this activity is the Communicational Dimension.
- Represent Event-Plan Connections activity: this aims at modeling the synchronization and the relationships between plans and events with activity diagrams extended for agent-oriented systems. The result of this activity is the Dynamic Dimension.

The approaches presented both in Kolp et al. (2005) and Wautelet et al. (2005) do not present an explicit and systematic way for selecting and applying the social patterns during design of multi-agent systems. The patterns are applied in an ad hoc way and their choice is not clearly justified.

4.4 Summary

In this chapter we presented the main limitations in related work regarding techniques to describe, use and implement the social patterns. In particular, we have discussed the lack of expressiveness and platform dependence of the current notation used to describe social patterns in Tropos. The crosscutting nature of a specific social pattern, the Matchmaker pattern, has been presented to argue for the need to consider the separation of social patterns concerns in MAS design. Moreover, we have pointed out some deficiencies in the current process to perform detailed design in Tropos. In fact, that process does not provide detailed activities to guide the design using the notation proposed by Kolp et al. (2005). Furthermore, it fails to provide a guide to select and systematically apply the social pattern in the detailed design of MAS.

In this context, this thesis defines an agency metamodel (by extending the UML metamodel) which captures several concepts and relationships to specify MAS. According to this agency metamodel, we define six UML-based diagrams to capture six views of MAS design: architectural, communication, environmental, intentional,

rational and plan. From these diagrams, currently we use the architectural and communication diagrams to develop a technique to describe social patterns using advanced separation of concerns in MAS detailed design. Moreover, we define a process to guide the use of both our agent-oriented modeling notation and the social patterns described using our technique. Next chapter presents the agency metamodel and the agent-oriented modeling notation to be used in MAS design.

5 Agency Metamodel and Notation

Agents provide developers with a flexible way to structure systems around autonomous, communicating elements. To support the efficient development of such systems, design techniques need to be introduced. In this context, we propose an extension of the UML 2.0 metamodel to support agency features and UML-based diagrams which can be used to capture six views of multi-agent systems architecture (Architectural, Intentional, Environmental, Communication, Rational and Plan).

Agents offer a new and often more appropriate manner to develop complex systems, which execute in open and dynamic environments. To support the development of such systems, tools and techniques need to be introduced such as methodologies to guide analysis and design, and proper abstractions to enable developers to deal with the complexity of agent-oriented systems [Luck et al., 2003].

Tropos [Castro et al., 2002] [Giorgini et al., 2005] relies on the i^* notation to describe both requirements and architectural design. However, the use of i^* as an architectural description language (ADL) for MAS is not suitable, since it has some limitations to express some information required for designing MAS architectures, such as interaction protocols, sequence of actions to perform a plan, rights to access resources in the environment, beliefs, and so on. To address this issue, we present an approach wherein an extended version of UML 2.0 is used as a notation to describe MAS architecture in Tropos. This proposal includes (i) an agency metamodel [Silva, C. et al., 2006c], which defines the constructs required to specify structural and dynamic features of MAS according to the Belief-Desire-Intention model [Rao and Georgeff, 1995] and FIPA standards [FIPA, 2004]; (ii) six views of MAS architectural design modeled by six UML-based diagrams [Silva, C. et al., 2006c] [Silva, C. et al., 2006d]; (iii) a set of guidelines to help the specification of MAS according to those diagrams [Silva, C. et al., 2006b] [Silva, C. et al., 2006c] [Silva, C. et al., 2006d].

The rest of this chapter is organized as follows. Section 5.1 presents the agency metamodel. Section 5.2 shows the agency profile. Section 5.3 describes the modeling diagrams based on the agency metamodel. Section 5.4 discusses related works. Finally, section 5.5 summarizes this chapter.

5.1 An Agency Metamodel

Aiming at providing a notation which supports MAS specification at the architectural level, in Silva, C. et al. (2006c) we have decided to extend the UML metamodel to support agency concepts. Although there is not much consensus yet in the literature

regarding the properties required to specify MAS, the Agency metamodel has been motivated by our previous work [Silva, C. et al., 2004] [Silva, C. et al., 2006b], where we have established some agent properties required to specify MAS. The concepts and relationships presented in the Agency metamodel are inspired by the Tropos ontology [Yu, 1995] but aims at being a generic proposal, to be used by other agent-oriented methodologies. For the sake of simplicity, the metamodel defining the agency features is divided into two categories: intentional and interaction. The intentional category concepts are described in Figure 5.1 while the interaction category concepts are described in Figure 5.2.

In the intentional category, a MAS can be conceived as an Organization [Ferber, 1999], which is composed of a number of AgentRoles as well as of other Organizations. The AgentRole concept extends the UML metaclass Class from the StructuredClasses package which extends the metaclass Class (from the Kernel package) with the capability of having an internal structure and ports. Norms are required for the Organization to operate harmoniously and safely. They define a policy and constraints that all the organizational members must comply with [Minsky and Muarata, 2004]. The Organization is typically immersed in exactly one Environment that the Agents may need to interact with to play their AgentRoles [Silva, V. et al., 2005]. An AgentRole has *rights* to access resources which belong to the environment [Zambonelli et al., 2003]. The Right metaclass possesses four boolean properties: *create*, *destroy*, *read* and *write*. The *Agent*, *Organization*, *Norm*, *Environment* and *Resource* concepts extend the UML metaclass *Class*, since they represent information that needs to be encapsulated into a class.

Each Agent can play one or more AgentRoles, and an Agent which plays an AgentRole has the Intention to achieve the AgentRole's goals. An agent commits itself to achieve goals and to execute plans. Thus, the Intention metaclass has the *commit* property. Based on Brazier et al. (2001) we consider two commitment strategies, defined as the enumeration class Commitment: (i) *single-minded*: an agent may drop commitments when it believes they can no longer be attained, regardless of changes in its goals; (ii) *open-minded*: an agent may drop commitments when it believes they can no longer be

attained or when the relevant goals are no longer desired. After committing to a goal and an associated plan, an agent starts the plan realization. The *Right* and *Intention* concepts are extensions of the UML metaclass *AssociationClass*, since they represent information that appear just because there is a relationship between two other elements and that information needs to be encapsulated into a class.

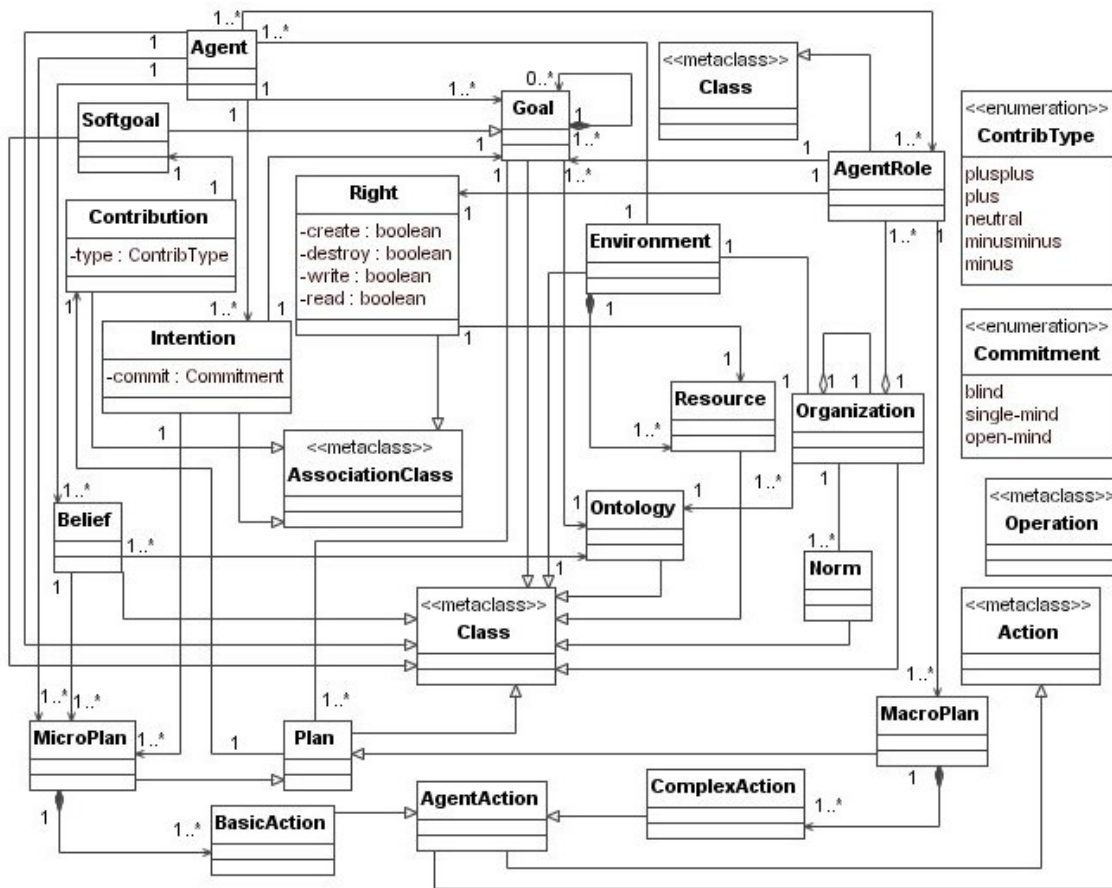


Figure 5.1 – Agency metamodel reflecting intentional concepts

A Goal is a condition or state of affairs in the world that the Agent has committed itself to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered [Mylopoulos et al., 2001]. A Plan encapsulates a recipe for achieving some goal. A softgoal is a condition or state of affairs in the world that the Agent would like to achieve, but unlike in the concept of (hard) goal, there are levels of achievement. These concepts extend the UML metaclass *Class*, since they represent information that needs to

be encapsulated into a class. Plans can contribute positively (+ or ++), negatively (- or --) or have no influence in the satisfaction of a softgoal. That influence of a plan in the achievement of a softgoal is called contribution. The *Contribution* concept is an extension of the UML metaclass *AssociationClass*, since it represents information that appears in the relationship between a plan and a softgoal and, therefore, needs to be encapsulated into a class. The contribution can be of five types: plusplus, plus, neutral, minunsmminus and minus, defined as the enumeration class *ContribType*.

An *AgentAction* determines the steps to perform a plan and extends both the *Action* and *Operation* UML metaclasses. The *AgentAction* has two subclasses: the *ComplexAction*, which can be further refined and the *BasicAction*, which cannot be decomposed. A *Plan* has two subclasses: a *MacroPlan* if the *Plan* is defined by the *AgentRole* and a *MicroPlan* if the *Plan* is defined by the *Agent*. The difference between *MacroPlan* and *MicroPlan* is that the former is a partial plan and, therefore, is composed of *ComplexActions*, and the latter is the complete and final plan and, therefore, is composed of *Basic Actions*. *Beliefs* represent the information the *Agent* has about itself and its current environment [Wooldridge, 2002]. These are preconditions for both executing its plans and ordering the plans' actions (i.e., the *MicroPlans*). These concepts extend the UML metaclass *Class*, since they represent information that needs to be encapsulated into a class. The description of both *Beliefs* and *Goals* must comply with the *Ontology* used in the *Organization*, i.e., the vocabulary of concepts which belongs to the system domain. We define the *MAS Ontology* by extending the UML metaclass *Class*. The rationale is that the UML class diagram has been employed to describe ontology in *MAS* [Silva, I., 2005].

Since *AgentRoles* are going to be used in the system architectural design, we will define them by using the organizational architectural features defined in Castro et al. (2003). These features, defining the interaction category, are depicted in Figure 5.2 and include: *OrganizationalPort*, *AgentConnector*, *Dependum*, *Dependee*, *Depender* and *AgentConnectorEnd*. They extend respectively the UML metaclasses *Port*, *Connector*, *Interface*, *InterfaceRealization*, *Usage* and *ConnectorEnd*.

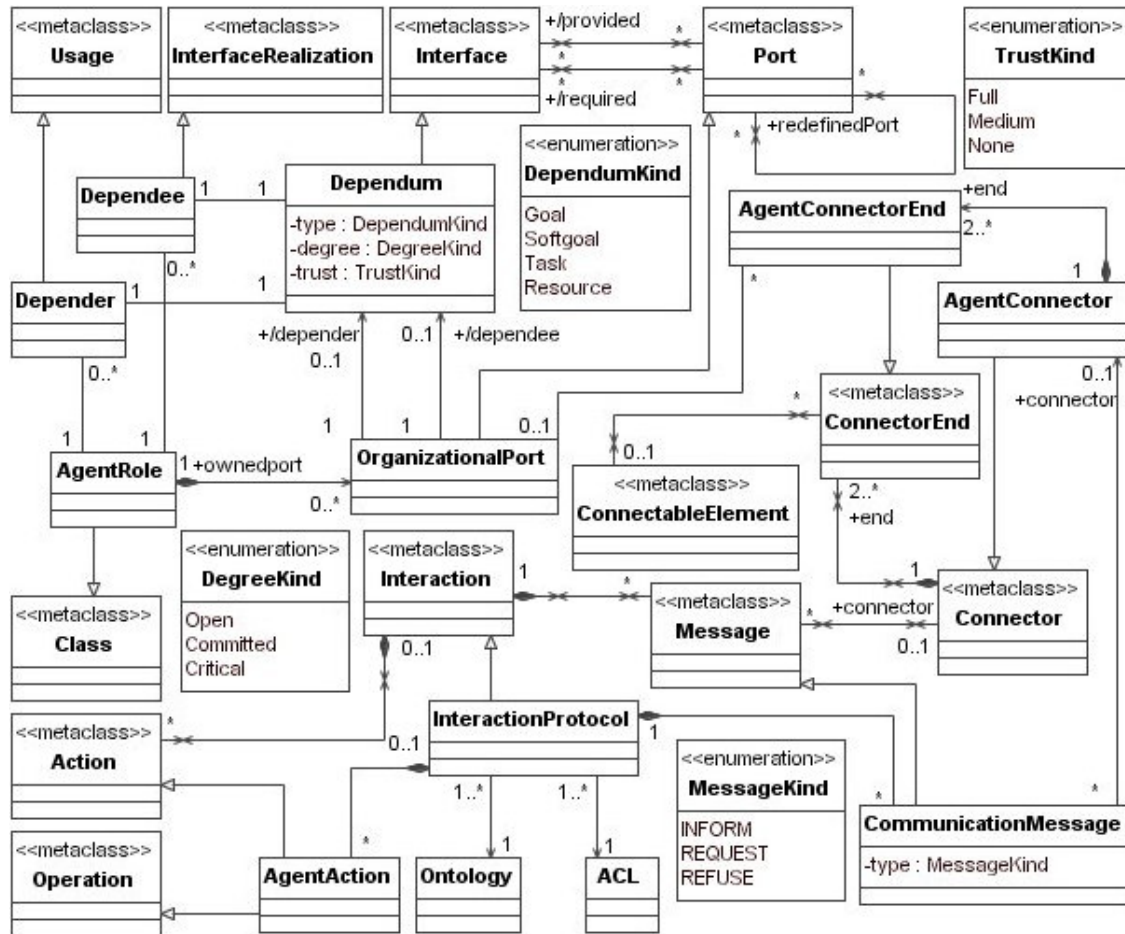


Figure 5.2 – Agency metamodel reflecting interaction concepts

A Dependum defines an “agreement” of service providing between two AgentRoles which play the roles of Depender and Dependee. Thus, the AgentRole responsible for providing the service possesses an OrganizationalPort playing the role of dependee and is related to the Dependum through a Dependee relationship. The AgentRole which requests the service possesses an OrganizationalPort playing the role of depender and is related to the Dependum through a Depender relationship. A dependum can be of four types: goals, softgoals, tasks and resources [Yu, 1995], defined as the enumeration class DependumKind. When an AgentRole depends on another AgentRole to achieve a (soft)goal (to fulfill a task or to deliver a resource), it is implicitly intended that the AgentRole trusts the other AgentRole and delegates it for such activities [Susi et al., 2005]. Thus, the *Dependum* metaclass has the *trust* property, which can be of three types:

full, medium and none, as defined in the enumeration class *TrustKind*. A dependency between two *AgentRoles* possesses different degrees, defined as the enumeration class *DegreeKind*: (i) *open*, the depender is affected without serious consequences when the achievement of the dependum fails; (ii) *committed*, the depender is significantly affected when the achievement of the dependum fails, and; (iii) *critical*, the depender is seriously affected when the achievement of the dependum fails [Yu, 1995].

AgentRoles need to exchange signals through an *AgentConnector* to accomplish the contractual agreement of service providing between the *AgentRoles*. An *OrganizationalPort* specifies a distinct interaction point between the *AgentRole* and its environment. An *AgentConnectorEnd* is an endpoint of an *Agentconnector*, which attaches the *AgentConnector* to an *OrganizationalPort*. Each *AgentRole* can interact with other *AgentRoles* according to an *InteractionProtocol*. An *InteractionProtocol* describes a sequence of *CommunicationMessages* that can be sent or received by Agents playing certain *AgentRoles* through the execution of their *AgentActions*. In addition, the *InteractionProtocol* must comply with an *Ontology* and an *Agent Communication Language (ACL)*. The *Ontology* is the vocabulary of the terms used in the message contents and their meaning (both the sender and the receiver must ascribe the same meaning to symbols for the communication to be effective). The *ACL* defines the format of each *CommunicationMessage* which composes the *InteractionProtocol*. The *ACL* concept extends the UML metaclass *Class*, while the *InteractionProtocol* concept extends the UML metaclass *Interaction*. The *CommunicationMessage* concept extends the UML metaclass *Message* and can be of several types including *REQUEST*, *INFORM* and *REFUSE*, among other performatives defined by the FIPA (2004) (defined as the enumeration class *MessageKind*). These indicate what the sender intends to achieve by sending the message.

5.1.1 Agency Profile

A Profile has been defined in the UML 2.0 specification to give a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a

particular domain, platform, or method. For example, in our approach we have created an agency metamodel by extending some UML metaclasses to address agent-oriented concepts. To enable MAS modeling by using UML constructs and tools, we use the profile mechanism to adapt the UML metamodel with constructs that are specific to the agent paradigm according to the agency metamodel defined in Figure 5.1 and Figure 5.2. Such adaptation is grouped in a profile, called *Agency Profile*. An extension (a kind of association) is used to indicate that the properties of a metaclass are extended through a stereotype.

Figure 5.3 presents some extensions we have made according to the agency metamodel defined in Figures 5.1 and 5.2, such as the stereotype *Agent* extending the UML metaclass *Class*, the stereotype *OrganizationalPort* extending the UML metaclass *Port*, the stereotype *Dependum* extending the UML metaclass *Interface*, the stereotype *Depender* extending the UML metaclass *Usage*, the stereotype *Dependee* extending the UML metaclass *InterfaceRealization* and the stereotype *AgentConnector* extending the UML metaclass *Connector*.

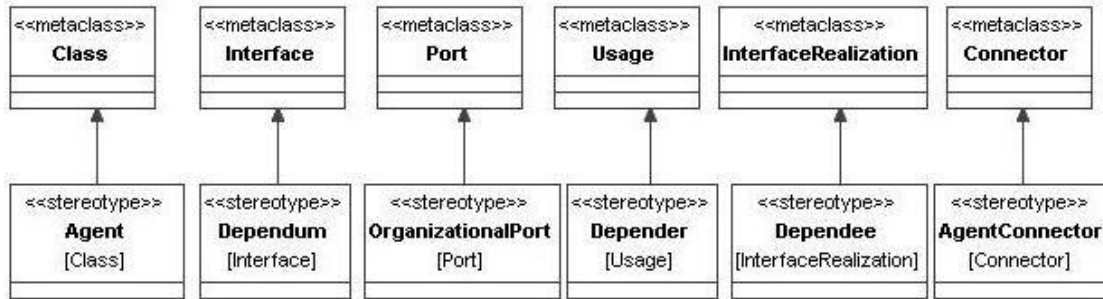


Figure 5.3 – Agency stereotypes

5.2 Agent-Oriented Modeling

In this section, we present the MAS modeling diagrams specified according to our agency metamodel. These diagrams were conceived to model six views of MAS design: Architectural, Communication, Environmental, Intentional, Rational and Action.

5.2.1 Architectural diagram

The architectural diagram extends the class diagram [OMG, 2005b], which is used to model the static view of modules that compose a system. In particular, the architectural diagram (Figure 5.4) reflects the client-server pattern [Shaw and Garlan, 1993] tailored for MAS. It is defined in terms of AgentRoles that possess goals achievable by plans. Since an Agent playing some AgentRole is not omnipotent, it needs to interact with other Agents (also playing AgentRoles) in order to accomplish its responsibilities. An AgentRole possesses OrganizationalPorts which enable the exchange of messages with other agents through AgentConnectors to accomplish some Dependium (i.e., service contract). For example, Figure 5.4 shows the Provider AgentRole, responsible for performing the service defined in the Dependium. This AgentRole aims at achieving the ServicePerformed goal by executing the PerformPlan MacroPlan, which, in turn, consists of performing the service() ComplexAction.

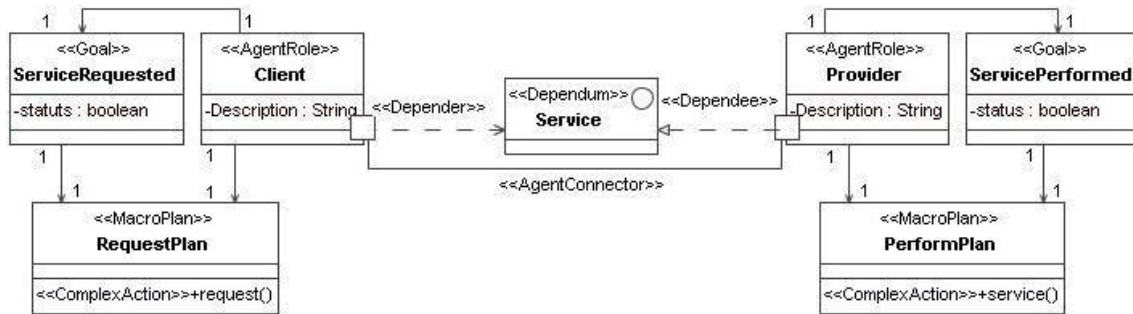


Figure 5.4 – MAS Architectural Diagram

The Client AgentRole aims at achieving the ServiceRequest goal by executing the RequestPlan MacroPlan, which, in turn, consists of performing the request() ComplexAction. Therefore, the Client AgentRole is responsible for requesting the service defined in the Dependium. Both the message for requesting the service execution and the message for confirming whether the service was successfully concluded are sent through the AgentConnector. Information such as *type:DependumKind*, *degree:DegreeKind* and *trust:TrustKind* are part of the Dependium specification (similarly to the *isAbstract:Boolean* property which is part of the Class specification in UML 2.0) and

cannot be graphically modeled. This information could be added to the model element through tagged values [OMG, 2005b], which are used to define model element's properties which are not predefined in UML.

5.2.2 Communication diagram

The communication diagram extends the sequence diagram [OMG, 2005b], which is used to model the communication patterns of a set of entities as they interact to implement behavior. The communication diagram is defined in terms of Agents playing given AgentRoles and the messages exchanged between them to achieve the provision of a service. For example, Figure 5.5 shows an interaction involving the agents playing the Client and Provider AgentRoles. The interaction specified using the communication diagram is asynchronous. The expression *client : Client* indicates an Agent *client* which plays the *Client* AgentRole. Information such as *Ontology* and *ACL* cannot be graphically modeled since they are part of the InteractionProtocol specification.

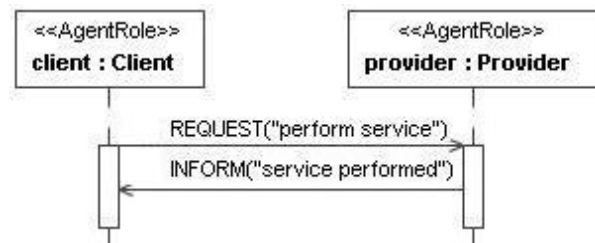


Figure 5.5 – MAS Communication Diagram

5.2.3 Environmental diagram

The environmental diagram extends the class diagram [OMG, 2005b] to model agentRoles comprising an organization which is situated in an environment. This environment is composed of resources, which are accessed by the AgentRoles according to their rights in order to accomplish their responsibilities. For example, Figure 5.6 shows the *Provider* AgentRole part of the *Org* organization which is situated in the *Env* environment.

The Provider AgentRole needs to access a *Res* resource available in the *Env* environment to fulfil its responsibilities. The *Provider* AgentRole can only read the *Res* resource, according to its *P-R Access* right (read *Provider-Res Access* right). Information such as *create:Boolean*, *destroy:Boolean*, *read:Boolean* and *write: Boolean* cannot be graphically modeled since it is part of the Right specification.

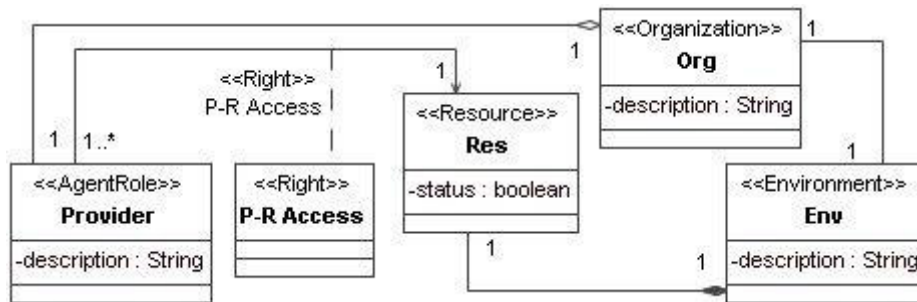


Figure 5.6 – MAS Environmental Diagram

5.2.4 Intentional diagram

The intentional diagram extends the class diagram [OMG, 2005b] to model agent roles, agents, their beliefs, goals, plans, as well as the norms and the ontology used in the organization. For example, Figure 5.7 shows the *Provider* AgentRole composing the *Org* organization which must comply with the *OrganizationalNorm* norm. The *AgentY* Agent, which plays the *Provider* AgentRole, believes that some request message has been received (depicted as *RequestReceived* belief). Information such as *commit:CommitmentKind* cannot be graphically modeled since it is part of the Intention specification.

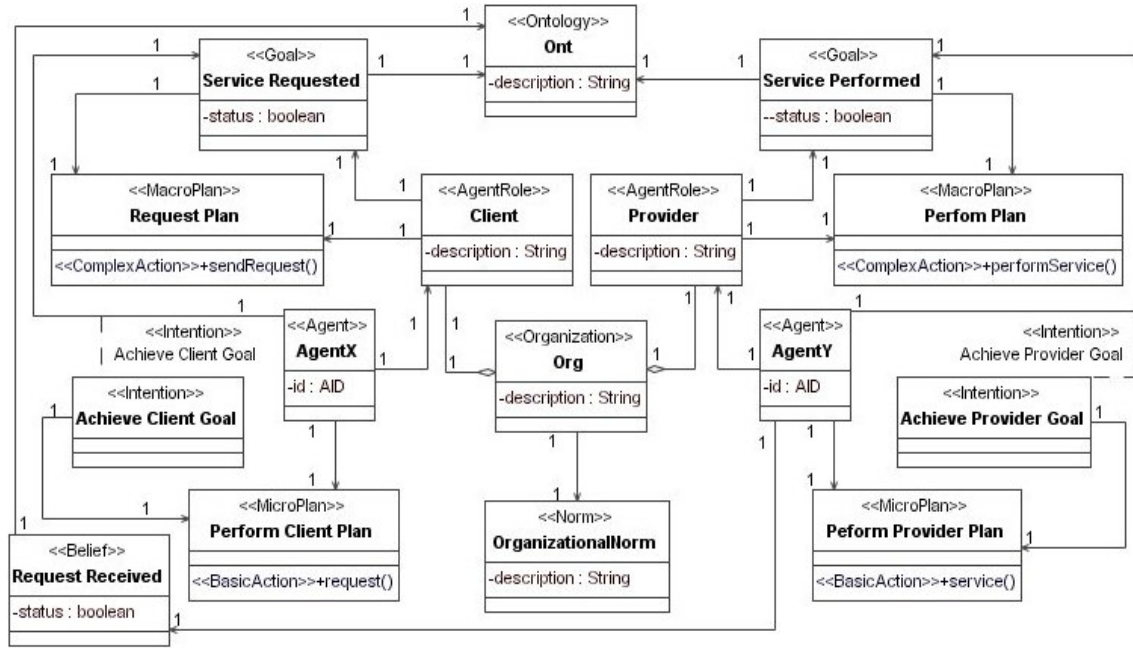


Figure 5.7 – MAS Intentional Diagram

5.2.5 Rational diagram

The rational diagram (Figure 5.8) extends the class diagram [OMG, 2005b] to model softgoals, the contribution of plans to satisfy softgoals, goals, plans to achieve those goals and agent roles.

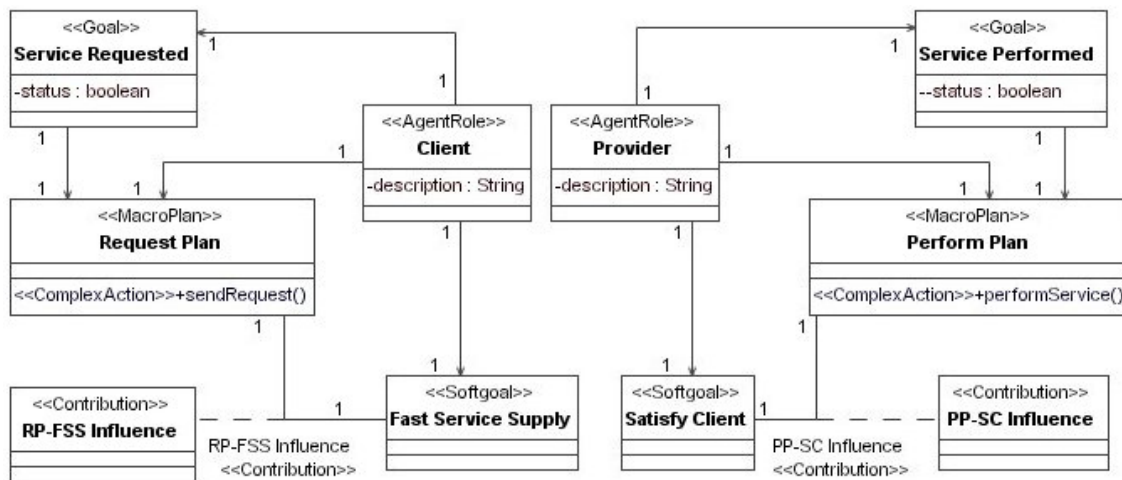


Figure 5.8 – MAS Rational Diagram

For example, Figure 5.8 shows the *Fast Service Supply* softgoal that *an agent* playing the *Client* agentRole is responsible for satisfying it. The *Client* agentRole has the *Request Plan* macroPlan to achieve the *Service Requested* goal. This plan has some influence in the satisfaction of the *Fast Service Supply* softgoal, which is captured in the *RP-FSS Influence* contribution (read *Request Plan-Fast Service Supply Influence* contribution). This influence is part of the Contribution specification, stored in the type: ContribType meta-attribute, and cannot be graphically modeled. Capturing that influence in the satisfaction of softgoals may help the choice of the most suitable plan to achieve a certain goal.

5.2.6 Plan diagram

The plan diagram (Figure 5.9) extends the activity diagram [OMG, 2005b], which is used to model the sequence of actions which compose a plan to achieve a goal. Through this diagram we can see which actions can be performed in parallel and which actions are dependent upon other actions. Capturing this information is essential to understand how an agent can work to achieve a goal. This diagram can be used to model the sequence of actions into macroPlans (composed of complexActions and outlined by the agentRole) or microPlans (composed of basicActions and defined by the agent). An extension of the activity diagram has also been used in the proposal of Penserini et al. (2006) to model these dynamic aspects of an agent.

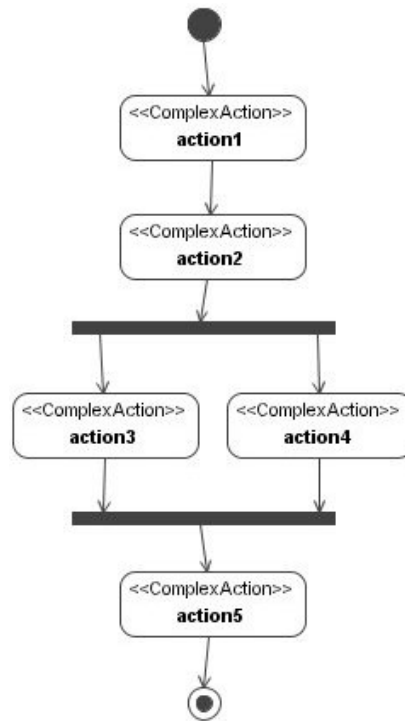


Figure 5.9 – MAS Plan Diagram

5.3 Comparison with Related Work

Guizzardi and Wagner (2005) presents an agent ontology based on the unified foundational ontology (UFO) and shows how it can be used as a foundation of agent concepts and for evaluating agent-oriented modeling methods. UFO is stratified into three ontological layers: (i) its core, UFO-A; (ii) the perdurant (i.e. process) extension layer, UFO-B; and (iii) the agent extension layer, UFO-C. Although this work provides a foundation for conceptual modeling, including agent-oriented modeling, it does not propose a modeling language for MAS based on this ontology. However, the proposed ontology can be used as a type of ‘mirror’ for our modeling language, i.e. for verifying how clear and expressive our language is.

Guizzardi (2006) proposed an agent-oriented approach named ARKnowD (Agent-oriented Recipe for Knowledge Management Systems Development) to guide the creation and evolution of knowledge management solutions. It has extended the UFO-C to create an ontology to evaluate, adjust and combine the notations adopted in Tropos

[Giorgini et al., 2005] and AORML [Wagner, 2003] to develop knowledge management information systems. The main limitation of this approach is that some Tropos concepts are not addressed by the ARKnowD's language, such as the concepts of softgoal and contribution. Furthermore, that notation does not address the concept of norm.

Several other languages for MAS modeling have been proposed in the last few years, such as AUML [Odell et al., 2000], MAS-ML [Silva and Lucena, 2004], SKwyRL-ADL [Mouratidis et al., 2005] and AORML [Wagner, 2003]. We discuss them next.

The work presented in Mouratidis et al. (2005) proposes a metamodel which defines an ADL to specify secure MAS. In particular, SKwyRL-ADL includes an agent, a security and an architectural model and aims at describing secure MAS, more specifically those based on the BDI model. Moreover, the Z specification language is used to formally describe SkwyRL-ADL concepts. Our proposal also supports MAS specification according to the BDI model. Furthermore, our notation supports architectural features, such as ports, connectors, interfaces and protocols and provides diagrams which enable the specification of MAS in six views. Our approach also provides a guide to specify design using the proposed diagrams (to be presented in chapter 7). Since we have both extended an agency metamodel from UML 2.0 metamodel and created an agency profile, we can use the UML constructs and tools to model MAS architectural design.

The proposal of a multi-agent system modeling language called MAS-ML is presented in Silva and Lucena (2004). It extends the UML metamodel according to the TAO (Taming Agents and Objects) metamodel concepts [Silva, V. et al., 2003]. TAO provides an ontology that defines the static and dynamic aspects of MAS. The MAS-ML includes three structural diagrams – Class, Organization and Role diagrams – which depict all elements and all relationships defined in TAO. The sequence diagram represents the dynamic interaction between the elements that compose a MAS — i.e., between objects, agents, organizations and environments. Although MAS-ML is an extension of UML metamodel, we couldn't use it in the detailed design of Tropos because it does not capture the concepts of i* framework, which are used from early requirements to

architectural design phases in Tropos. Thus, compared with that approach, our proposal is compatible with Tropos framework, supports architectural features, such as ports, connectors, interfaces and protocols, as well as its use is guided by some heuristics.

AUML [Odell et al., 2000] provides extensions of UML, including representation in three layers of agent interaction protocols, which describes the sequence of messages exchanged by agents as well as the constraints in messages content. However, AUML does not provide extensions to capture the agent's reasoning mechanisms (individual structure) or the agent's organization (system structure). On the other hand, we provide UML-based diagrams to capture the agent internal structure and the MAS structure, as well as a guide to use these diagrams in MAS specification.

In AORML [Wagner, 2003], an entity is either an agent, an event, an action, a claim, a commitment, or an ordinary object. Commitments and claims establish a special type of relationship between agents. Besides human and artificial agents, it also models institutional agents. Institutional agents are usually composed of a number of human, artificial, or other institutional agents that act on its behalf. It models communicative action events (or messages), non-communicative action events, action events and non-action events. There are two basic types of AORML models: external and internal models. External models describe the perspective of an external observer who is watching the agents and their interactions, and include: Agent Diagrams, Interaction Frame Diagrams, Interaction Sequence Diagrams and Interaction Pattern Diagrams. Internal models adopt the view of a particular agent to be modeled and include: Reaction Frame Diagram, Reaction Sequence Diagram and Reaction Pattern Diagram. Currently AORML does not include the concept of goals neither allows modeling the proactive behavior of agents, which is based on action planning and plan execution for achieving goals. On the other hand, our approach supports MAS specification according to the BDI agent model [Rao and Georgeff, 1995] which defines proactive behavior of agents.

In summary, we are concerned with the detailed specification of MAS design by providing a standard notation which captures six views of MAS architecture and

heuristics to guide the use of this notation (to be presented in chapter 7). Furthermore, our approach explicitly models the purpose - resource delivery, task performing or (soft)goal achievement - associated to an interaction between two agent roles. This information is derived from actor dependency in i^* models. Our approach is being developed in the context of the Tropos framework, aiming at supporting all phases of the MAS development lifecycle.

5.4 Summary

The approach presented in this chapter aims to address current issues related to the lack of proper techniques and notation to specify MAS architectural design. In particular, this notation is also needed to specify detailed design and describe social patterns. We have proposed both the agency metamodel and the UML-based diagrams to capture six views of MAS at architectural level and, therefore, support specification of MAS architectural design in more detail. Besides supporting the BDI model and FIPA standards, the proposed diagrams also address MAS architectural constructs (ports, connectors, interfaces and protocols) and constructs such as agent role, intention, commitment, trust, agent communication language and degree of dependency. Finally, these diagrams may be used in other frameworks/methodologies, facilitating the development and design of MAS.

To describe social patterns using advanced separation of concerns, in the next chapter, we present a technique called Agent Pattern Specifications (APS). In doing so, it is recommended to use a consistent template, besides graphical models, because in doing so we can document decisions, concrete examples as well as benefits and costs analysis which promoted its use. Thus our approach to describe social patterns currently uses a template and two graphical models to describe the architectural and communication views of the pattern. This description technique promotes separation of social patterns concerns from MAS concerns in detailed design.

6 Agent Pattern Specifications

This chapter proposes an approach to support separation and modularization of crosscutting concerns in multi-agent systems. Aspect-Oriented Software Development offers mechanisms to encapsulate crosscutting concerns in separate modules, the aspects. Aspects are used as abstractions to capture social patterns concerns that cut across functional modules in MAS. To achieve this, we propose a technique to describe social patterns in an aspect-oriented context.

Tropos has defined a set of design patterns, called social patterns [Kolp et al., 2005], focusing on social and intentional features which are recurrent in cooperative and multi-agent systems. Although both application core and design patterns concerns are independent from each other, the current use of patterns can lead to the scattering and tangling of their concerns within the application functional modules. As a consequence, the application core becomes dependent on patterns [Noda and Kishi, 2001], which do not fully meet quality requirements, such as understandability, maintainability, evolvability and reusability. Thus, design patterns concerns can be called crosscutting concerns, since they cut across the functional modules concerns, decreasing the system reusability and maintainability. These crosscutting concerns can be better addressed by adopting aspect-oriented software development techniques [Kiczales et al., 1997]. The purpose of this new technology is to localize crosscutting concerns, avoiding their scattering or tangling through software artifacts [Elrad et al., 2001].

Although many languages for multi-agent systems modeling have been proposed in the last few years, such as AUML [Odell et al., 2000] and MAS-ML [Silva and Lucena, 2004], they do not support Tropos concepts properly nor consider the separation of crosscutting concerns. On the other hand, some languages for modeling separation of crosscutting concerns have been proposed, such as aSideML [Chavez, 2004] and AODM [Stein, 2002], but they do not address MAS directly. In this context, we propose a technique called APS [Silva, C. et al., 2007a] to describe social patterns using abstractions and mechanisms provided by aspect orientation to support the separation of crosscutting concerns in MAS. To achieve this, we need to: (i) specialize the agency metamodel [Silva, C. et al., 2006c] by using the concept of model roles [Kim et al., 2003] used in Pattern Specifications technique [France et al., 2004]; and (ii) attach notes to the structural model of the social pattern to capture some abstractions of aspect orientation.

This chapter is organised as follows: Section 6.1 introduces our notation to describe social patterns in an aspect-oriented context. Section 6.2 suggests guidelines to map the proposed notation to the constructs of AspectJ and JADE environments. Section 6.3 presents some related work. Finally, section 6.4 summarises the chapter.

6.1 A Technique for Social Pattern Specification

To describe social patterns we propose an approach that specializes the agency metamodel for MAS architectural diagram (as described in both Chapter 5 and Silva, C. et al. (2006c)) by using the concept of model roles [Kim et al., 2002]. In fact, the concept of model roles was used in the Pattern Specifications technique [France et al., 2004] to specialize the UML metamodel for specifying which model elements must participate in a pattern. The work presented in Araújo et al. (2004) uses the Pattern Specifications technique [France et al., 2004] to address aspect modeling. Analogously, we can use PSs to promote the separation of social patterns concerns in MAS design [Silva, C. et al., 2007a].

The following sub-sections describe the template used to specify a social pattern (section 6.1.1), the respective structural specification (section 6.1.2) and communication specification (section 6.1.3). These are illustrated by the Matchmaker pattern.

6.1.1 Pattern Template

Here we will use a subset of the GoF's template [Gamma et al., 1994] to describe the social patterns. A complete example of the Matchmaker social pattern template is depicted in Table 6.1.

Table 6.1 – The Template for a Pattern Description

Element	Description
<i>Name</i>	Matchmaker Pattern
<i>Intent</i>	To locate a provider for a given service requested by a client and letting the client directly interact with the provider.
<i>Applicability</i>	When an agent (client) needs to directly interact with another agent (provider) to use its services, but does not know which agent offers the desired service.
<i>Motivation Example</i>	A reporter agent needs to interact with a news agency to obtain news about a specific subject. The news agency to be contacted is going to be known at run time by any reporter agent.

	known at run time by any reporter agent.
<i>Problem</i>	How can clients locate unknown providers which offer a specific service?
<i>Solution</i>	The solution involves an intermediary agent (matchmaker) that receives requests from service providers to subscribe/unsubscribe its services into the yellow pages maintained by it. An agent (client) may need a specific service provided by an unknown agent (provider). The Matchmaker also receives requests from client agents to locate some provider agent which offers a specific service. If there is some provider for the requested service, the Matchmaker gives that provider's identification to the client which, in its turn, can directly interact with it.
<i>Participants</i>	The Client requests the identification of an agent that provides a specific service. The Matchmaker finds the Provider of the requested service and gives its identification to the Client. The Provider must subscribe the Matchmaker yellow page service in order to be found by the clients requesting its services.

6.1.2 Structural Agent Pattern Specification

A Structural Agent Pattern Specification (SAPS) defines the part of the pattern metamodel that characterizes MAS architectural diagram views of pattern solutions. It defines subtypes of agency metaclasses describing MAS architectural diagram elements (e.g., agency metaclasses *AgentRole*, *AgentConnector*) and specifies semantic pattern properties using constraint templates (see Figure 6.1). An SAPS consists of a structure of pattern roles [France et al., 2004], where a role specifies properties that a MAS model element must have if it is to be part of a pattern solution model. Formally, a role defines a subtype of an agency metaclass. The metaclass is called the base of the role. For example, a role that has the metaclass *AgentConnector* as its base specifies a subset of MAS agent connectors. A MAS model element conforms to (or plays) a role if it satisfies the properties defined in the role, that is, the element is an instance of the subtype defined by the role. In an SAPS a role can be a classifier or a relationship role. A role that has the base *Classifier* or a base that is a subtype of *Classifier* (e.g.,

AgentRole, Dependum) is a classifier role. A relationship role is any role that has the base Relationship or a base that is a subtype of Relationship (e.g., AgentConnector).

Figure 6.1 shows an SAPS that specifies solutions for the Matchmaker pattern. In the Matchmaker pattern, a matchmaker agent locates a provider corresponding to a client request for service, and then hands the client a direct handle to the chosen provider. The SAPS for the Matchmaker pattern is described using the client-server architectural pattern (Figure 5.4). The Matchmaker pattern has to provide three services (i.e. dependums): *Locate Provider*, *Perform Service* and *Subscribe Yellow Pages*.

For example, the shaded area of Figure 6.1 represents the classes involved in the *|Locate Provider* service defined in a Dependum class. When the *|Client* AgentRole executes the *Request Provider Identification* MacroPlan, by performing the *requestProviderID* ComplexAction to achieve the *Provider Identified* Goal, it triggers a request to the *|Matchmaker* AgentRole to perform the *|Locate Provider* Service. The *|Matchmaker* AgentRole performs the requested service because it does not conflict with the achievement of the *Yellow Pages Service Provided* Goal. A conflict is detected when the service requested to an agent playing a specific agent role, can cause the failure of some of its goals. So, both the requested service and the goal achievement are accomplished by means of the *Yellow Pages Server* MacroPlan. The description of the *|Perform Service* and *Subscribe Yellow Pages* services is achieved in a similar way.

Here, we use the idea proposed in Araújo et al. (2004) to allow an SAPS to contain both role elements and non-role (i.e. concrete modeling) elements. Thus, we use model roles only in the elements of the pattern that will vary from one application to another. When the pattern is applied to a specific application, these roles will be instantiated to concrete modeling elements of the application. The inclusion of concrete modeling elements in the pattern specifications reduces the number of instantiation steps since in large systems can present a great number of model roles instantiations and, therefore, break the scalability of the approach.

The SAPS in Figure 6.1 consists of five model roles in the classes: three for AgentRole metaclass (Client, Provider and Matchmaker) and two for Dependum metaclass (*|Locate Provider* and *|Perform Service*). The roles define subtypes (specializations) of metaclasses in the Agency metamodel. The Client role defines a subtype of metaclass

AgentRole in the Agency metamodel. The AgentRole roles indicate that the appropriate architectural diagrams must have at least one AgentRole that conforms to the Client role (as indicated by the 1..* realization multiplicity in the first compartment of the AgentRole role), at least one AgentRole that conforms to the Provider role, and at least one AgentRole that conforms to the Matchmaker role.

An AgentRole that conforms to the Matchmaker role (referred to as a Matchmaker class) must have exactly one goal that conforms to the Yellow Pages Service Provided role and exactly one MacroPlan that conforms to the Yellow Pages Server role. The same rationale can be applied to the other roles which are subtypes of metaclass AgentRole in the Agency metamodel.

Furthermore, the SAPS presents in the notes attached to the AgentRole classes three composed model roles: |Matchmaker.lplan.laction, |Client.lplan.laction and |Provider.lplan.laction. Observe that some of these composed model roles are repeated in the notes. It describes the AgentRoles' points affected by the aspects encapsulating the Matchmaker pattern's concerns (each aspect will encapsulate the concern of one pattern participant). For example, let us consider the note attached to the |Client AgentRole. It represents that the execution of the requestProviderID(), which is part of the Request Provider Identification, is going to affect the client participant of the Matchmaker pattern before the execution of some of its actions, which is part of some of its plans. To determine who the client participant is in the application and its affected action, we need to instantiate each model role present in the SAPS (e.g., |Client, laction, lplan, etc.) to design elements present in the MAS architectural design.

Similarly, the upper note attached to the |Provider AgentRole represents that the execution of the unsubscribe(), which is part of the Register Services, is going to affect the provider participant of the Matchmaker pattern after the execution of some of its actions, which is part of some of its plans. The lower note attached to the |Provider AgentRole represents that the execution of the subscribe(), which is part of the RegisterServices, is going to affect the provider participant before the execution of some of its actions, which is part of some of its plans. To determine who are the provider participant in the application and its affected action, we need to instantiate each model role present in the SAPS (e.g., |Provider, laction, lplan, etc.) to design elements present in the MAS architectural design.

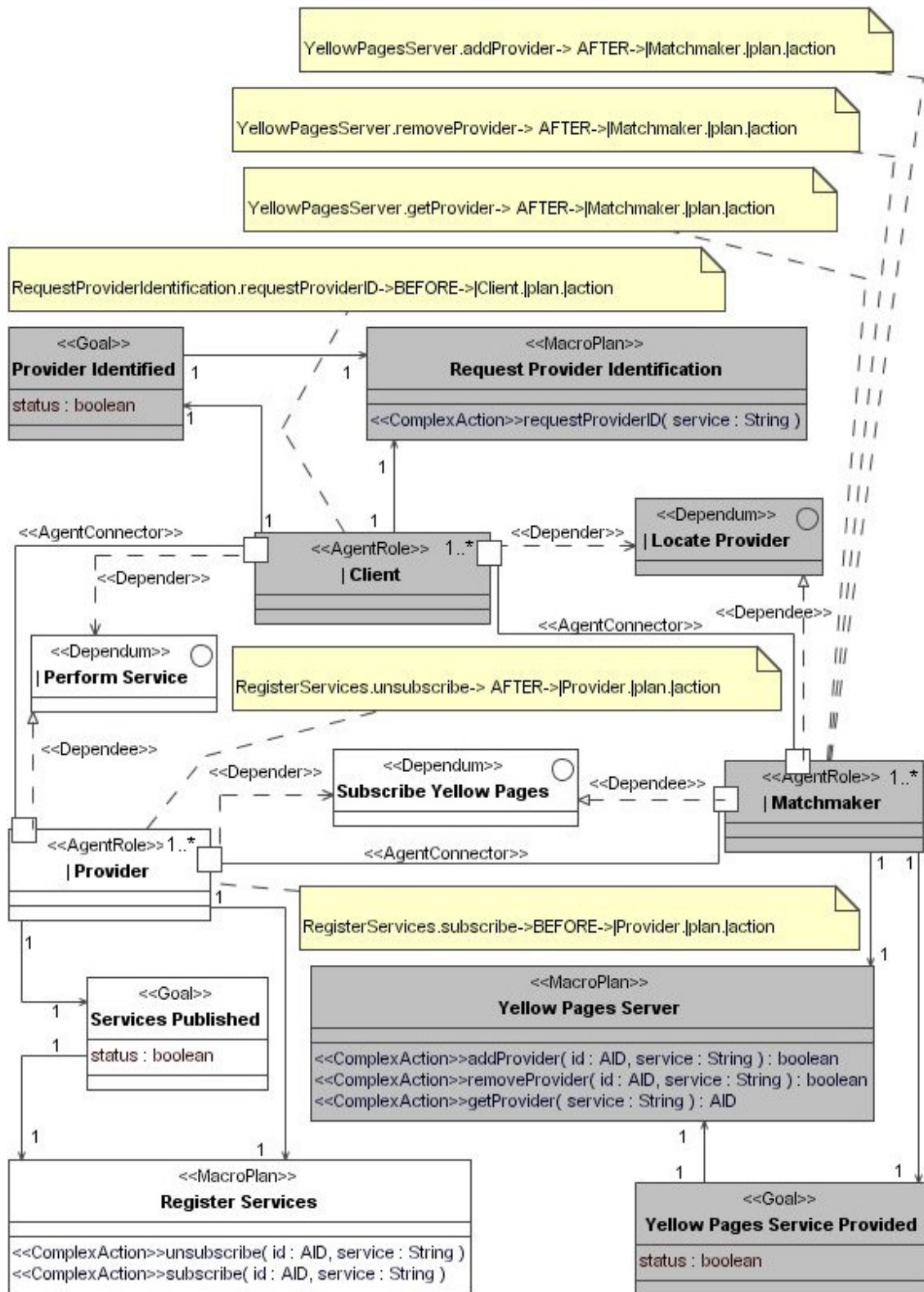


Figure 6.1 – Matchmaker Structural Agent Pattern Specification

Finally, the upper note attached to the **|Matchmaker** AgentRole represents that the execution of the **addProvider()**, which is part of the **Yellow Pages Server**, is going to affect the matchmaker participant of the Matchmaker pattern after the execution of

some of its actions, which is part of some of its plans. The central note attached to the `lMatchmaker AgentRole` represents that the execution of the `removeProvider()`, which is part of the Yellow Pages Server, is going to affect the matchmaker participant of the Matchmaker pattern after the execution of some of its actions, which is part of some of its plans. The lower note attached to the `lMatchmaker AgentRole` represents that the execution of the `getProvider()`, which is part of the Yellow Pages Server, is going to affect the matchmaker participant of the Matchmaker pattern after the execution of some of its actions, which is part of some of its plans. To determine who are the matchmaker participant in the application and its affected action, we need to instantiate each model role present in the SAPS (e.g., `lMatchmaker`, `laction`, `lplan`, etc.) to design elements present in the MAS architectural design. Thus, we weave the aspects with the `AgentRoles` resulting in the application of the pattern to the system.

Next section presents the technique to capture the pattern of communications of the agents involved in the Matchmaker pattern.

6.1.3 Communication Agent Pattern Specification

A Communication Agent Pattern Specification (CAPS) describes a pattern of communications and is defined in terms of model roles defined in an SAPS. The SAPS model roles are used to specify participants in a communication pattern. Formally, a CAPS defines a part of the pattern metamodel that specifies conforming communication diagrams. A CAPS defines a specialization of the *InteractionProtocol* Agency metaclass. Since *InteractionProtocol* extends the *Interaction* class from UML 2.0, it is a structure of *Lifelines* and *CommunicationMessages* (which extends the *Message* class from UML 2.0). Consequently, an *interactionProtocol* role is a structure of *lifeline* and *communicationMessage* roles. Each *lifeline* role is associated with an *agentRole* role: a participant that plays a *lifeline* role is an agent which plays an *agentRole* that conforms to the *agentRole* role. *Lifeline* roles define specializations of the *Lifeline* class and *communicationMessage* roles define specializations of *CommunicationMessage* class.

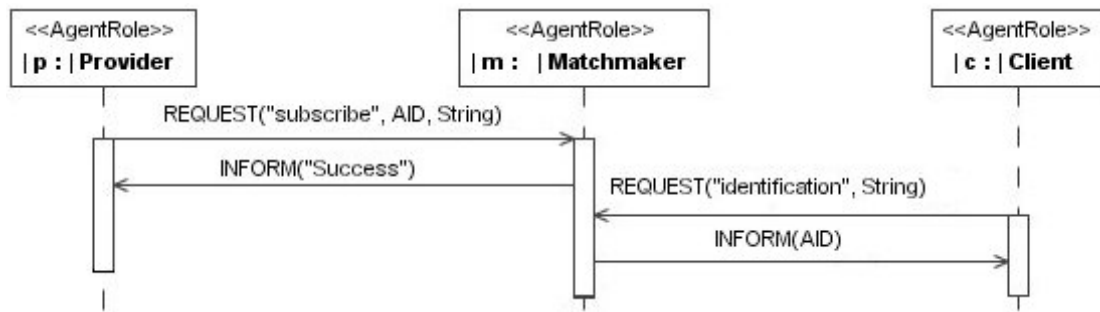


Figure 6.2 – Matchmaker Communication Agent Pattern Specification

Figure 6.2 shows a CAPS that describes the pattern of communications between a *Client*, a *Matchmaker* and a *Provider*. The expression $lc : |Client$ indicates that an agent, conforms to the lifeline role lc and plays the agentRole that, in turn, conforms to the agentRole role $|Client$ defined in the Matchmaker SAPS.

A communication diagram conforms to a CAPS if the conforming communications respect the format of the exchanged messages specified in the CAPS. Each message contains a FIPA performative and data. The communication diagram shown in Figure 6.2 depicts the main scenario of the communication pattern for the participants involved in the Matchmaker pattern: the $|Provider$ sends a REQUEST message to subscribe its data (by sending its identification, typed by AID and its service description, typed by String) in the Matchmaker’s Yellow Pages. The $|Matchmaker$ sends an INFORM message which contains the request result (e.g., “success”, for the primary scenario). The $|Client$ sends a REQUEST message to get the identification of some service provider (by sending the description of a service, typed by String). Finally, the $|Matchmaker$ sends an INFORM message which contains the search result (e.g. the provider identification, for the primary scenario).

Next we propose some mapping guidelines to enable the implementation of social patterns concerns separately from agents concerns through the integration of JADE and AspectJ implementation environments.

6.2 Towards Implementation in JADE and AspectJ

To implement our approach we propose the integration of the JADE and AspectJ environments. This integration is not difficult as both implementation environments are based on Java, facilitating the combination of JADE and AspectJ code in the same program. We could use JADEX [Braubach et al., 2004] since it is a software framework

for the creation of goal-oriented agents following the BDI model [Rao and Georgeff, 1995]. However, currently our purpose is illustrating the feasibility of using aspects to implement social patterns in MAS, regardless of the deliberative behaviour of the software agents. To this end, using JADE is sufficient and simpler than JADEX. Notice that the JADEX reasoning engine is realized on top of the JADE platform and therefore, in the future, we can integrate JADEX and AspectJ.

This section provides some guidelines to map our notation to represent MAS architecture (Figure 5.4) and social patterns (Figure 6.1) in constructs of JADE and AspectJ, respectively. With respect to the affected Architectural Diagram: (i) each «MacroPlan» becomes a Behavior in JADE; (ii) each «ComplexAction» becomes a common method; (iii) each «AgentRole» becomes an Agent in JADE, since JADE does not support the concept of agent roles played by agents; (iv) the goal element is not mapped to any JADE construct, as JADE does not support the implementation of cognitive agents [Silva, C. et al., 2007a].

With respect to the SAPS: (i) each «MacroPlan» presented in Figure 6.1 becomes an Aspect in AspectJ; (ii) each «ComplexAction» becomes an Advice in the aspect it belongs to; (iii) each «AgentRole» is a module affected by an aspect; (iv) each note attached to an AgentRole becomes a Pointcut and also defines the rule to compose the aspect with the agent; (v) the goal is not mapped to any AspectJ construct.

6.3 Comparison with Related Work

Most of the design patterns proposed in agent-oriented community deal with patterns for designing mobile agent-based applications, such as Tahara et al. (1999), Aridor and Lange (1998) and Deugo et al. (1999). Others propose a catalogue of coordination patterns for MAS [Hayden and Yang, 1999], or a social perspective on agent-oriented design patterns [Kolp et al., 2005]. Most of these approaches (except the last one), do not use a standard notation to describe the patterns and none of them provide a systematic strategy to select and apply patterns in the detailed design of MAS.

The approach presented in Kolp et al. (2005) proposes a framework to explore five different complementary dimensions required to describe both social patterns and detailed design models. These dimensions are social, intentional, structural, communicational and dynamic. The social dimension is described using i^* [Yu, 1995];

the intentional dimension is described using a table; the last structural, communicational and dynamic dimensions are described using extensions of UML [OMG, 2005b]. The UML's class diagram is extended to support the design of BDI agent architecture [Rao and Georgeff, 1995] and JACK [AOS, 2005] implementation. Therefore, the work also supports generation of code from agent's specifications to JACK platform. During the design phase it is expected that the specification of MAS be as detailed as possible to serve as a guide to direct implementation in any available platform (not only JACK). The models produced in the detailed design stage of our approach can be implemented in any FIPA (2004) compliant platform which supports the BDI agent model [Rao and Georgeff, 1995]. Moreover, the approach of Kolp et al. (2005) does not address separation of design patterns concerns.

The approach of Quenum et al (2003) focuses on protocol engineering and deals with the coupling between interaction features and functional features which places a limitation on the reusability potential of the agent's interaction protocol. In particular, it proposes that instead of designing the interaction model by hand, the agent programmer builds it by using the functional model to configure and specialize generic coordination protocols from a public library that the agent system contains. By separating the interaction and functional models, there can be flexible updates in the agent's model. Our approach deals with the coupling between social patterns features and functional features which mainly fails the reusability of both functional modules and social patterns implementation. In particular, we propose to separate the social patterns and functional features by using advanced separation of concerns principles. Since social patterns features include both functional and interaction features to be applied to a MAS, our approach is a step forward in comparison to the Quenum et al. (2003) approach.

Aspect-oriented modeling requires the use of a higher-level aspect model that addresses the aspect-oriented programming concepts at a preliminary design stage (avoiding language specific details), and allows the designer to work at a more abstract level during software construction and evolution. Aspect-Oriented Design Model (AODM) [Stein, 2002] enhances the existing UML specification with aspect-oriented concepts that reproduces the crosscutting characteristics of the AspectJ language. The aSideML [Chavez, 2004] is an aspect-oriented modeling language based on the UML that provides notation, semantics and rules for specifying aspects and crosscutting at the

design level of OO systems. Composition patterns [Clarke and Walker, 2001] is an approach that handles crosscutting concerns at design level by means of templates. However, these approaches do not address MAS directly.

It is argued in Garcia (2004) that the design and implementation of agent internal architecture concerns (e.g. interaction, adaptation, autonomy, knowledge, collaboration, roles, learning and mobility) tend to affect or crosscut many classes of the system design and code, including those representing the basic agent functionality. To address this issue, Garcia (2004) proposes an aspect-oriented approach to support the separate handling and modularization of MAS specific concerns. The proposed approach encourages the separate handling of agent properties, and provides a disciplined scheme for their composition. In our approach, we are not concerned with the separation of agent properties since we consider the agents already built-in (composed of several agent properties, such as autonomy, interaction, learning, etc.) and supported by some agent implementation platform. In fact, we are interested in using aspect-oriented abstractions and mechanisms to support the separation and modularization of social patterns concerns in MAS and its latter composition with the agents composing the system.

In Silva, C. et al. (2006a), we have extended the aSideML [Chavez, 2004] to incorporate agency features to be used in both the separation of social patterns concerns and its latter weaving with the system agents. However, this approach becomes somewhat complex because we extended the aSideML with agency features, architectural features and model roles. On the other hand, our current proposal is simpler than the previous one because we have extended the UML metamodel with agency features and specialized the agency metamodel by using model roles.

6.4 Summary

In this chapter we have proposed a notation for specifying social patterns using advanced separation of concerns. To this end, we have defined a technique called APS to describe the social patterns concerns separately from MAS functional modules (i.e. agent roles). Since the APS technique is based on a standard modeling language (UML), the social patterns could be easily used by several agent-oriented methodologies as well as modeled by any UML CASE (Computer Aided Software Engineering) tool. In Silva,

C. et al. (2006f) we have used this standard notation to describe organizational architectural styles [Kolp et al., 2002].

We envisage several benefits for MAS design and implementation, such as improving concerns modularity, since social patterns concerns are localized into aspects while the application concerns are localized into agent roles. Consequently, this improves software reusability and maintainability. To illustrate the feasibility of implementing our approach, we have outlined a mapping from the proposed notation to an implementation environment integrating JADE [Bellifemine et al., 2003] and AspectJ [Kiczales et al., 2001]. The next chapter presents a process to perform detailed design which guides the design of MAS using our agent-oriented modeling language as well as the selection and systematic application of social patterns described according to the APS technique.

7 Detailed Design Process

In this chapter we outline a process to perform the detailed design of multi-agent systems. This process guides the description of MAS using a UML-based notation. It helps the selection of the proper social pattern(s) and provides a systematic way to apply social patterns. The Detailed Design Process is described using the Software Process Engineering Metamodel (SPEM) notation.

The importance of the software development process (henceforth referred to as “process”) is based on the fact that it contributes to improve software quality and satisfy stakeholders, minimizes development cost and time, and provides support for future improvements [Sommerville, 2004]. However, the definition of a process is not an easy task [Pressman, 2003]: (1) processes are complex, i.e., highly iterative, with parallel tasks and non-linear relationships; (2) environments are always changing, since they present fast changes of technology or faces market pressures, as well as business requirements and human resources can change; (3) processes are either very rigid or they are not defined correctly. Having all this in mind, we will specify our process using a standard terminology, in particular the SPEM [OMG, 2006].

This chapter focuses on the detailed design phase of the software development lifecycle. It introduces additional detail to the structure and behavior of each architectural component of a system. In Tropos [Giorgini et al., 2005] the detailed design phase defines how the goals assigned to each actor present in the architectural model are fulfilled by agents performing roles according to social patterns [Kolp et al., 2005]. However, neither does Tropos provide a detailed description of these patterns, nor a systematic way to choose and apply them in the detailed design of the MAS under development. To solve these issues, we propose the definition of the Tropos Detailed Design Process [Silva, C. et al., 2007b] using the SPEM notation [OMG, 2006].

The rest of this chapter is organized as follows: Section 7.1 presents the Detailed Design process and describes it using the SPEM notation. Section 7.2 compares our proposal with existing related works. Finally, section 7.3 summarizes this chapter.

7.1 Detailed Design Process in SPEM

The detailed design phase of Tropos [Silva, C. et al., 2007b] involves two ProcessRoles (Agent Designer and Software Architect), three Guidances (Agency Metamodel, i* Framework and Mapping Heuristics) and ten WorkProducts (six UMLModels, two MASModels and three Documents), presented in Figure 7.1.

According to Figure 7.1, each ProcessRole performs specific activities that can use specific Guidances to consume and/or produce different WorkProducts. The *Agent Designer* ProcessRole is responsible for choosing the most appropriate agent model for the internal architecture [Ferber and Gutknecht, 2000]. The *Software Architect* ProcessRole is the person, team, or organization responsible for the systems' architecture [IEEE, 2000].

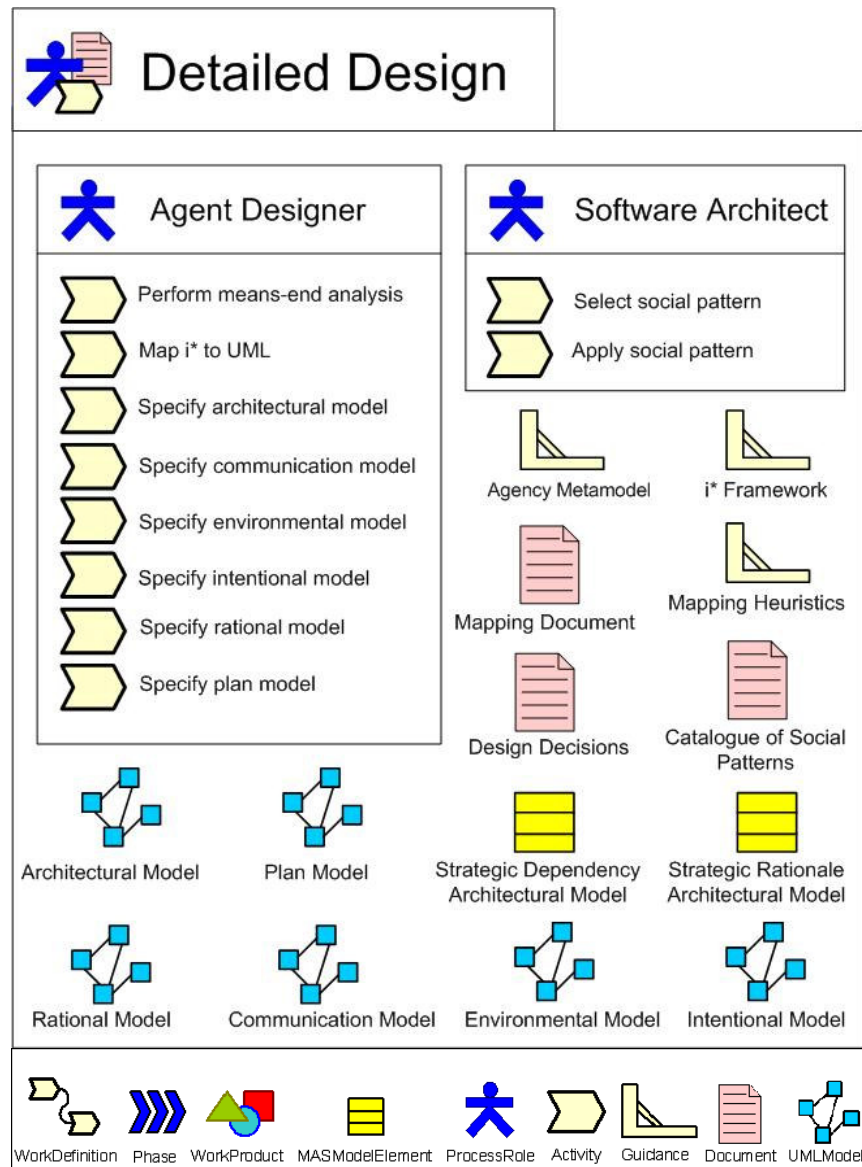


Figure 7.1 – Detailed Design Package

The flow of the *WorkDefinitions* in the detailed design phase is depicted in Figure 7.2. The first *WorkDefinition* is the *Agent Detailed Design*, which is in charge of producing the UML-based models from the i* architectural models. In doing so, it consumes the *SD*

Architectural Model WorkProduct and produces nine WorkProducts: *SR Architectural Model*, *Mapping Document*, *Architectural Model*, *Communication Model*, *Environmental Model*, *Intentional Model*, *Rational Model* and *Plan Model*. The second WorkDefinition is the *Social Pattern Selection*, which is in charge of selecting the proper social pattern to operationalize a given non-functional requirement (softgoal present in the SD architectural model). In doing so, it consumes the *Catalogue of Social Patterns* and the *SD Architectural Model* WorkProducts and produces the *Design Decisions* Document. If no social pattern has been selected then the Detailed Design Process is finished. Otherwise, the *Social Pattern Application* WorkDefinition is performed to apply the chosen social pattern and then update six WorkProducts: *Architectural Model*, *Communication Model*, *Environmental Model*, *Intentional Model*, *Rational Model* and *Plan Model*. If necessary, a new iteration is performed to select another social pattern. Otherwise, the Detailed Design Process is finished.

In Figure 7.3, the *Agent Designer* ProcessRole performs the *Agent Detailed Design* WorkDefinition which is composed of the following activities: *Perform means-end analysis*, *Map i^* to UML*, *Specify architectural model*, *Specify rational model*, *Specify plan model*, *Specify environmental model*, *Specify communication model*, *Specify intentional model*. The *Perform means-end analysis* and *Map i^* to UML* are performed in sequence to produce the *Mapping Document*. This workproduct is consumed by the following activities: *Specify architectural model*, *Specify rational model*, *Specify plan model*, *Specify environmental model*, *Specify communication model*, *Specify intentional model*. These activities are independent of each other and can be performed in parallel. When the *Agent Designer* finishes performing all these activities, the *Software Architect* can perform the *Social Pattern Selection* WorkDefinition which is composed of the *Select Social Patterns* activity. If some social pattern is selected, the *Software Architect* can perform the *Social Pattern Application* WorkDefinition which is composed of the *Apply Social Patterns* activity. This activity then applies the social pattern and updates the UML-based models.

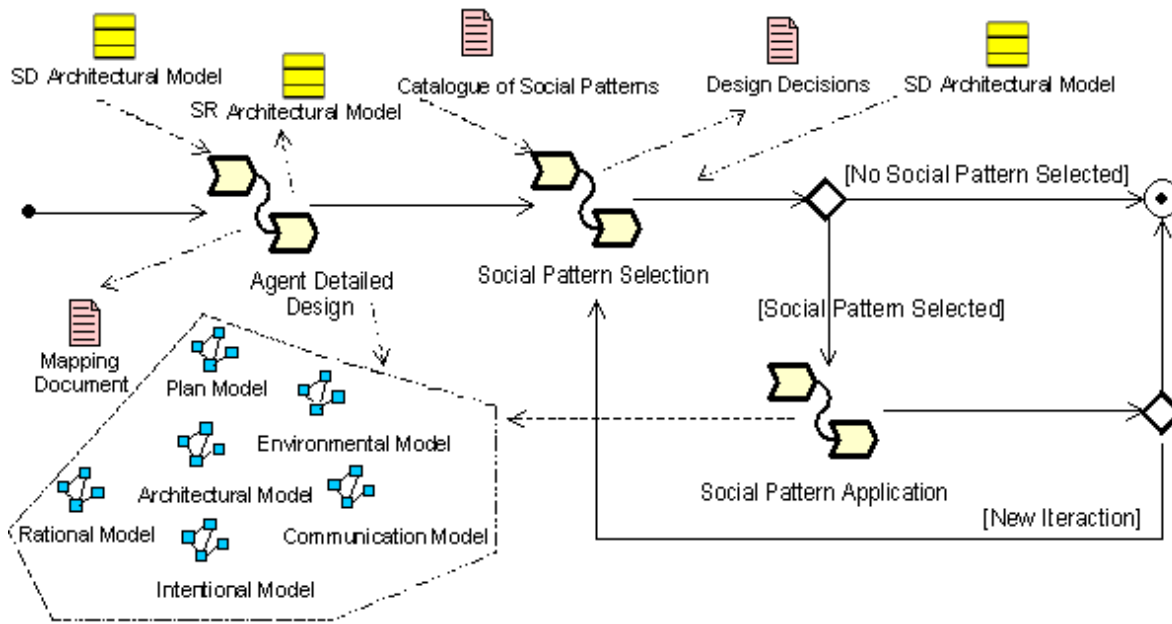


Figure 7.2 – WorkDefinitions Flow

In Figure 7.3, we can find the Activities performed by each ProcessRole, the WorkProducts consumed and produced by each activity as well as the workflow of the detailed design activities.

Perform means-end analysis helps identifying the reasons/motivations associated with each dependency that an agent possesses [Yu, 1995] and produces the SR Architectural Model WorkProduct. *Map i* to UML* uses a set of heuristics (see section 7.1.1) to guide the mapping of i* description of MAS architecture to the UML-based notation for MAS (see section 5.2) and produces the Mapping Document WorkProduct. *Specify Architectural Model* specifies the MAS according to the architectural diagram (aided by the Mapping Document WorkProduct) and produces the Architectural Model WorkProduct. *Specify Plan Model* specifies the MAS according to the plan diagram and produces the Plan Model WorkProduct. *Specify Rational Model* specifies the MAS according to the rational diagram and produces the Rational Model WorkProduct.

Specify Environmental Model specifies the MAS according to the environmental model (aided by the Mapping Document WorkProduct) and produces the *Environmental Model* WorkProduct. *Specify Communication Model* specifies the MAS according to the communication model (aided by the *Mapping Document* WorkProduct) and produces the

Communication Model WorkProduct. *Specify Intentional Model* specifies the MAS according to the intentional model (aided by the *Mapping Document* WorkProduct) and produces the *Intentional Model* WorkProduct. In *Select Social Pattern*, the software architect can, for example, analyze the templates (e.g. Table 6.1) that describe several features of each social pattern to address a specific requirement. This activity produces the *Design Decisions* WorkProduct.

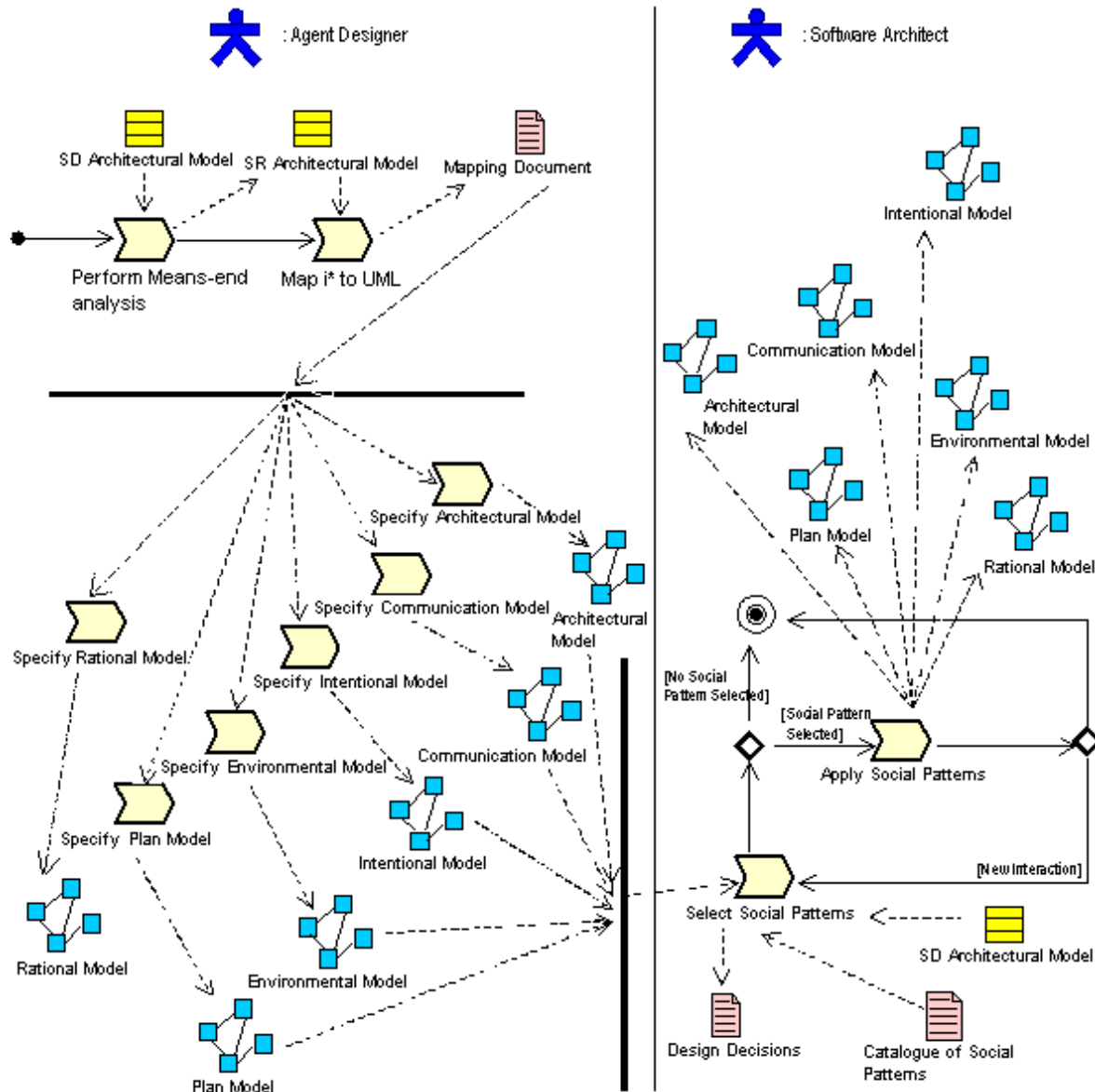


Figure 7.3 – Workflow for the Detailed Design Process

Finally, the *Apply Social Pattern* applies the chosen social patterns through instantiation of each model role present in the structure of the pattern to design elements present in the MAS architectural design (e.g., |Client binds to *Editor*, |Locate Provider bind to *Locate Photographer-Reporter*) and, therefore, updates the *Architectural*, *Environmental*, *Intentional*, *Communication*, *Rational* and *Plan Models*.

The *SD Architectural Model* is a WorkProduct consumed by the Perform means-end analysis activity. This is an i* framework model which describes a network of dependency relationships among various (architectural) actors, capturing the motivation and why of activities [Yu, 1995]. The *Catalogue of Social Patterns* is a WorkProduct consumed by the Select Social Pattern Activity and includes all the social patterns defined by Kolp et al. (2005) and described in detail using the the APS technique.

In the following we can find the explanation of each WorkProduct produced in the Detailed Design Process (Figure 7.2). The *SR Architectural Model* is an i* framework model which describes and supports the reasoning that each (architectural) actor goes through concerning its relationships with other (architectural) actors by using a means-ends analysis [Yu, 1995]. The *Architectural Model* is a UML-based model defined in terms of AgentRoles which possess goals achievable by plans. Since an AgentRole is not omnipotent, it needs to interact with other AgentRoles in order accomplish its responsibilities. An AgentRole possesses OrganizationalPorts which enable the exchange of messages with other AgentRoles through AgentConnectors in order to accomplish some Dependium (i.e., service contract) (see Section 5.2.1). The *Communication Model* is a UML-based model defined in terms of Agents playing AgentRoles and the messages exchanged between them to achieve a service providing (see Section 5.2.2).

The *Environmental Model* is a UML-based model defined in terms of AgentRoles composing an organization which is situated in an environment. This environment is composed of resources which are accessed by the AgentRoles according to their rights in order to accomplish their responsibilities (see Section 5.2.3). The *Intentional Model* is a UML-based model defined in terms of agents, their beliefs, agent roles, goals, plans, as well as the norms and the ontology used in the organization (see Section 5.2.4). The *Rational Model* is a UML-based model defined in terms of agent roles, softgoals,

contributions, plans and goals. The main objective of this model is to capture the influence of plan in the satisfaction of softgoals to help the choice of plan alternatives to achieve a goal (see Section 5.2.5). The *Plan Model* is a UML-based model described in terms of agent plans and actions (see Section 5.2.6) using an extension of the UML 2.0 activity diagrams [OMG, 2005b]. The *Mapping Document* stores textually the mapping between i* concepts and concepts of the UML-based notation for MAS. This document is going to be used to specify each of the UML-based models. The *Design Decisions* WorkProduct stores the decisions taken during the selection of social patterns for designing a MAS. This document is useful to keep tracing of what social pattern has been chosen to operationalize a given non-functional requirement. The *i* Framework*, *Agency Metamodel* and *Mapping Heuristics* are used by the *Agent Designer* ProcessRole to execute its activities.

The i* Framework includes concepts such as actor and social dependencies among actors as well as two diagrams: Strategic Dependency and Strategic Rationale [Yu, 1995]. The Agency Metamodel is an extension to the UML metamodel [OMG, 2005b] which defines a UML-based notation for MAS architecture (see Section 5.1). The Mapping Heuristics is a set of heuristics to describe MAS using our UML-based notation derived from an architectural description using i* (see Section 7.1.1).

Figure 7.4 presents the relationships between the *WorkProducts*, *Documents* and *Guidelines*. The Detailed Design Model *WorkProduct* is composed of the Architectural, Communication, Plan, Rational, Environmental and Intentional *UMLModels*.

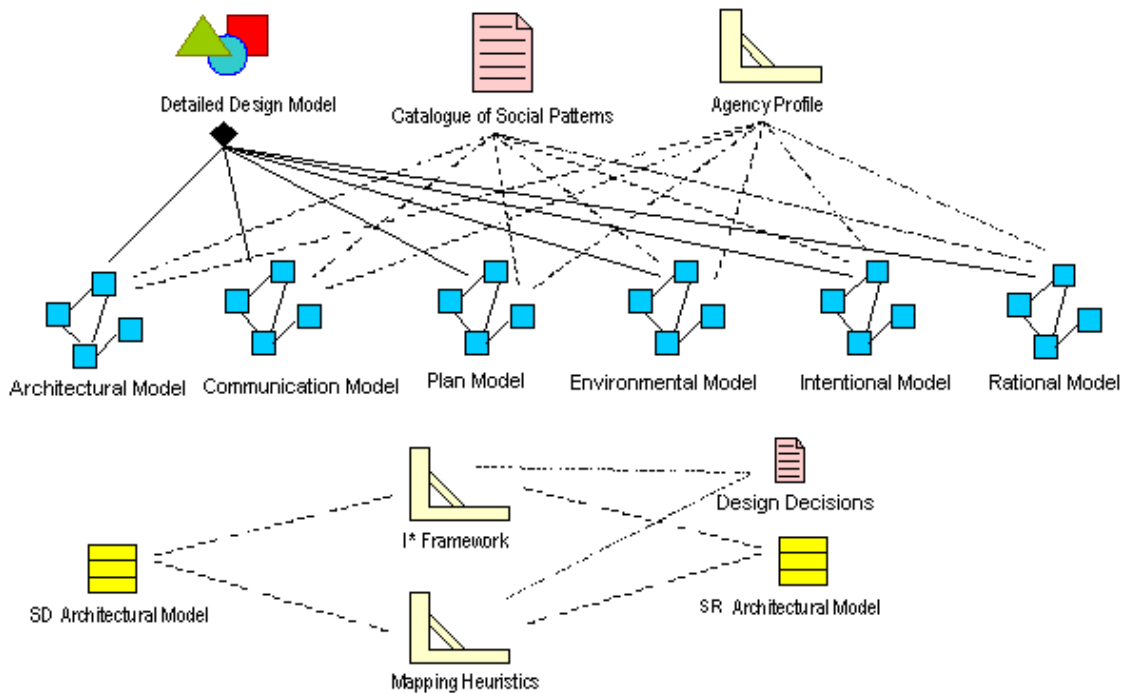


Figure 7.4 – Detailed Design WorkProducts Relationships

Next we can find the description of the Detailed Design WorkDefinitions presented in Figure 7.2.

7.1.1 Agent Detailed Design

To identify the information related to the notation presented in Chapter 5, it is necessary to execute the activity *Perform a means-end analysis* for each actor which belongs to the MAS architecture represented in i*. However, the process for performing the means-end analysis of these actors is a specialization of the original process [Yu, 1995]. We begin by establishing a main goal which can be decomposed into subgoals or operationalized by one or more tasks through the means-end link. Each task corresponding to a means element in some means-end relationships must be decomposed into other tasks, goals and resources. Subtasks cannot be further decomposed. The end element in the means-end relationship can only be a goal. Each goal must be operationalized by a means-end relationship or decomposed by a task-decomposition link. The means element in the means-end relationship can only be a task. Decomposed tasks can contribute positively (+ or ++), negatively (- or --) or have no influence in the achievement of softgoals.

After concluding the *Perform a means-end analysis* activity, we can start to perform the *Map i* to UML* activity. In Silva, C. et al. (2006b), we have presented some heuristics to map the i* concepts to both agency and UML-RT concepts [Selic and Rumbaugh, 1998]. However, at that stage we did not take into account the architectural concepts supported by UML 2.0. Hence, in Silva, C. et al. (2006c) we have redefined these heuristics to consider the MAS architectural concepts (Figure 5.4) extended from UML 2.0. Here, we define other heuristics to address the derivation of the new concepts we have added to agency metamodel. The *Map i* to UML* activity is guided by the following heuristics:

- H1. Each role in the i* model becomes an «AgentRole» class in the architectural diagram.
- H2. Each agent in the i* model becomes an «Agent» class in the intentional diagram.
- H3. Each play relationship between an agent and a role in the i* models becomes an association class relationship «Intention» between the correspondent «Agent» and «AgentRole» in the intentional diagram. Currently we do not provide guidelines to define the commit property of the «Intention». This is a subjective decision of the designer based on his domain knowledge.
- H4. Each dependum in the i* model becomes a «Dependum» interface in the architectural diagram. Note that a «Dependum» can be of four types (goals, softgoals, tasks and resources) according to the metamodel in Figure 5.2. These types are not provided explicitly in the model since it is a property of the model element.
- H5. In an i* model, a dependency is: (i) open if it has the symbol “o”; (ii) committed if it has no symbol, and; (iii) critical if it has the symbol “x”. Thus this information can be captured in the degree property of the «Dependum». Here guidelines to define trust of the «Dependum» are not provided.
- H6. Each depender in the i* model becomes a «Depender» usage in the architectural diagram.

- H7. Each dependee in the i* model becomes a «Dependee» interface realization in the architectural diagram.
- H8. Each dependency (depender -> dependum -> dependee) in the i* model becomes a «Connector» association in the architectural diagram. Ports are added to the agents to enable the link through the connector.
- H9. Each resource related to the actor in the i* model becomes a «Resource» in the environmental diagram. It represents an environmental resource which the agent needs to access to perform its responsibilities. Currently we do not provide guidelines to define the agent rights to access each «Resource». This is a subjective decision of the designer based on his domain knowledge.
- H10. Each goal (or softgoal) in the i* models which is not decomposed becomes a «Goal» in both the architectural diagram and intentional diagram. It represents the objectives the agent playing a specific role intends to achieve.
- H11. Each softgoal in the i* models which is not decomposed becomes a «Softgoal» in the rational diagram.
- H12. Each contribution link between a task and a softgoal becomes association class relationship «Contribution» between the correspondent «MacroPlan» and «Softgoal» in the rational diagram
- H13. Each task in the i* models becomes a «MacroPlan» in both the architectural diagram and intentional diagram. It represents the means through which a goal is going to be achieved.
- H14. Each sub-task in the i* models becomes a «ComplexAction» in the architectural diagram. It represents each step which composes a «MacroPlan».
- H15. A «Belief» is some condition for performing a task (i.e, a «Plan» or an «AgentAction»). It represents the knowledge the agent has about both the environment and itself.

- H16. The MAS «Organization» is composed of agent roles and other organizations.
- H17. Each (soft)goal that is part of a task becomes a «Goal» and creates a «ComplexAction» responsible for generating that goal in the «MacroPlan» correspondent to the task.

A preliminary mapping of i^* concepts into agent, goal, belief, plan and resource concepts was originally proposed in Castro et al. (2002). However, here we did not provide a guide or a notation to capture the mapped information. This thesis introduces a notation to be used in the MAS specification at the architectural level. Finally, we define a specialization of the original means-end analysis process which is appropriate for refining the rationale of each agent role before the mapping of the i^* concepts to the agency concepts.

After concluding the *Map i^* to UML* activity, we produce the Mapping Document which is consumed to produce the UML-based models (see Section 5.2) by performing each of the following activities: *Specify architectural model*, *Specify rational model*, *Specify plan model*, *Specify environmental model*, *Specify communication model*, *Specify intentional model*. As soon as the UML-based models have been specified, the *Software Architect* can perform the *Select Social Patterns* activity.

7.1.2 Social Patterns Selection and Application

The last steps of the process are selecting and applying social patterns to refine the architectural design of a multi-agent system. One of the key challenges is to choose the proper social pattern to be applied to a system architectural design. One can for example analyze the templates (e.g. Table 6.1) that describe several features of each social pattern to address a specific non-functional requirement. Analyzing the template of several patterns, we can conclude that the most suitable pattern to address a given requirement is a certain pattern or a combination of patterns.

To apply a social pattern we need to instantiate each model role (e.g., *IClient*, *ILocate Provider*) present in the structure of the pattern (Figure 6.1) to design elements present in the MAS architectural design (e.g., *Editor*, *Locate Photographer-Reporter*).

7.2 Comparison with Related Work

Methodologies such as GAIA [Zambonelli et al., 2003], Tropos and MaSE [DeLoach, 2006] have become the focus of the emerging area of agent-oriented software engineering. In GAIA, the detailed design phase is responsible for eventually identifying the agent model and the services model which, in turn, act as guidelines for the actual implementation of agents and their activities. The purpose of the design phase in MaSE is to take roles and tasks and to convert them into a form more amenable to implementation, namely agents and conversations. The MaSE design phase consists of four steps: designing agent classes, developing conversation, assembling agents and deploying the agents. Observe that all these methodologies do not support the application of design patterns for detailing the MAS architecture. Moreover, both GAIA and MaSE do not use a standard notation to specify the artifacts produced during the design phase.

The approach proposed by Quenum and Akinine (2005) focuses on interaction protocols in MAS, which determines pattern of collaborations between agents playing roles. In particular, they propose to select interaction protocols dynamically, where some interaction protocols can appear at runtime from safe and authenticated libraries of interaction protocols when needed. They argue that moving from static to dynamic protocol selection greatly increases the efficiency of the system and properly handles situations closer to openness where all the protocols are not known at design time. Our approach focuses on social patterns which include pattern of collaboration between agents and were proposed in the context of Tropos framework. Currently, Tropos does not support the development of open systems. Therefore, in this thesis, we are not concerned with dynamic selection of social patterns.

7.3 Summary

This work describes and formalizes the detailed design process using the SPEM notation. This process includes activities to map models initially described in the i^* notation to UML-based diagrams as well as to select and apply social patterns to the architectural design of MAS to make these applications more flexible, understandable, and reusable. The

results of this process are UML-based diagrams which capture five views of MAS (Architectural, Communication, Intentional, Environmental, Rational and Plan Diagrams).

Our detailed design process produces MAS models which can be implemented in any FIPA compliant platform [FIPA, 2004] which supports BDI agent model [Rao and Georgeff, 1995]. Moreover, the UML-based diagrams for MAS can be implemented using an integration of JADE [Bellifemine et al., 2003] and AspectJ [Kiczales et al., 2001] environments.

In next chapter, we present the first results of our approach using a case study to develop a Content Manager System [Silva, I., 2005], called e-News [Silva, C. et al., 2005], in the context of the Tropos framework.

8 Case Study: Newspaper Office

This chapter provides a case study to illustrate the feasibility of our proposal. In particular, we specify a multi-agent system for a Newspaper Content Management domain.

To illustrate our approach, let us consider the Electronic Newspaper example presented in Section 2.2.1. The e-News system (Figure 8.1) enables a user to read news by accessing the newspaper website maintained by a Webmaster AgentRole which is responsible for updating the published information. The information to be published is provided by the Chief Editor AgentRole. The Chief Editor depends on the Editor to have the news of a specific category. For example, an Editor may be responsible for political news, while another one may be responsible for sports news. Each Editor contacts one or more Photographers-Reporters to find the news of specific categories (e.g., sport news). The Chief Editor then edits the Editor' news and forwards them to the Webmaster to publish them. The early requirements, late requirements and architectural design of that system have been performed in detail as shown in the Appendices.

8.1 Detailed Design

The detailed design phase is intended to introduce additional detail for each architectural component of a system. It consists of defining how the goals, assigned to each actor present in the architectural model, are fulfilled by agent roles according to social patterns. The detailed design process introduced in Chapter 6 is going to be used in this phase to produce a detailed design model of a multi-agent system in the context of Tropos.

The e-News system (Figure 8.1) is composed of four AgentRoles: Editor, Webmaster, Chief Editor and Photographer-Reporter. The *Joint Venture* architectural style [Kolp et al., 2002] has been chosen and applied to the MAS architectural design. The reasons for that choice are presented in the Appendices. Here, our focus is on the design of the MAS architecture according to agent-oriented modeling diagrams (Chapter 5).

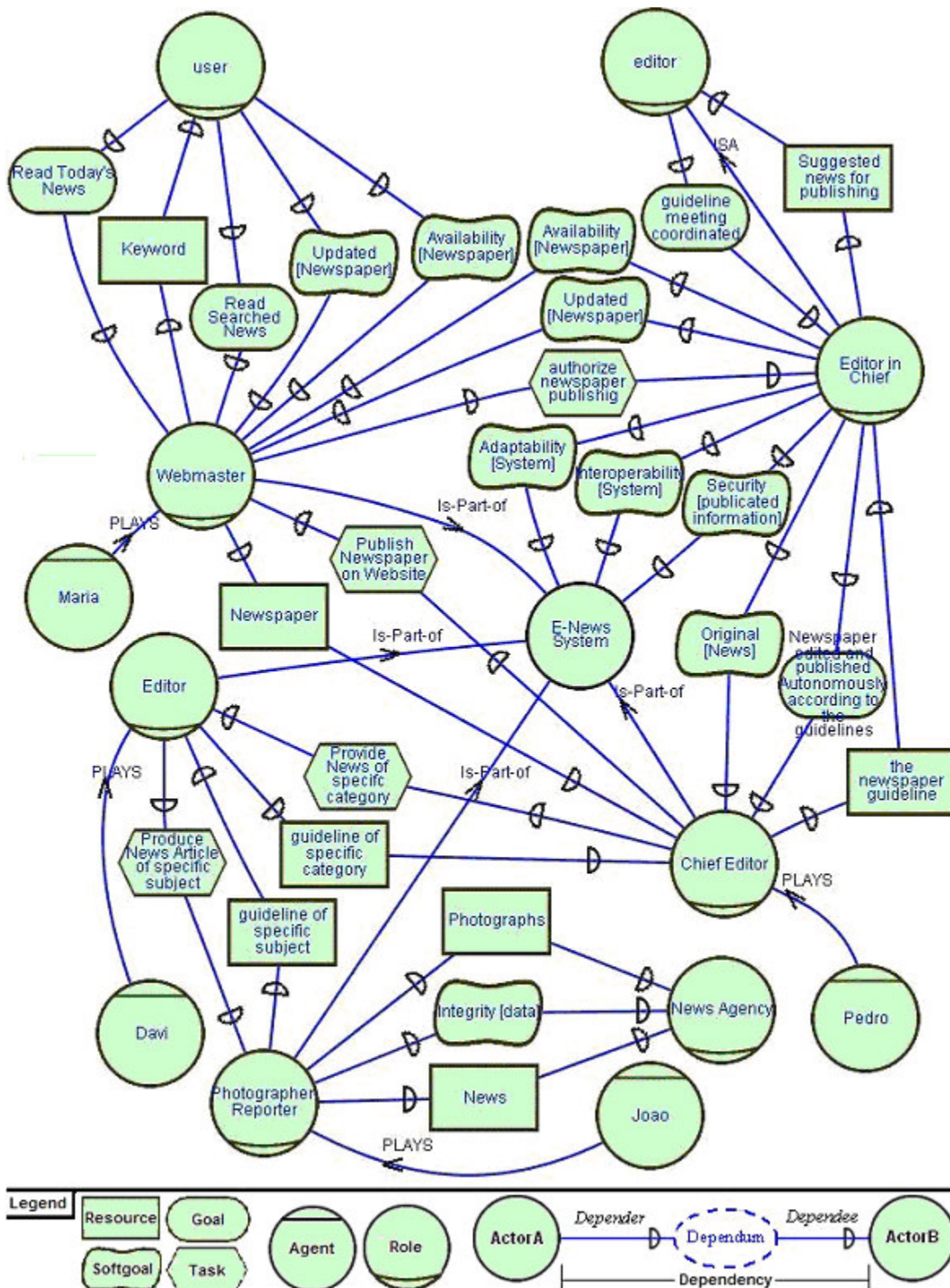


Figure 8.1 – e-News System Architecture

8.2 Agent Detailed Design

We start this activity by using the heuristics presented in Section 7.1.1 to produce MAS UML-based models at the architectural level in the context of Tropos. We begin by performing the means-end analysis for each actor which belongs to the MAS architecture described using the i^* notation. Then, we rely on the mapping heuristics to specify each diagram presented in Section 7.1.1.

For example, we perform the specialized means-end analysis of the Editor, Webmaster, Chief Editor and Photographer-Reporter actors in order to capture their rationale when pursuing their goals and dependencies.

The Editor expects to obtain *News of Specific Category Edited*. One alternative to satisfy this goal is to perform the *Edit News of Specific Category* task. This task is decomposed into five sub-tasks (see the refined model in Figure 8.2): *Format News Article*; *Select Unknown, Recent, Important and Accurate News*; *Review Photos Quality*; *Review News Content*; and *Provide Specific Subject Guideline*.

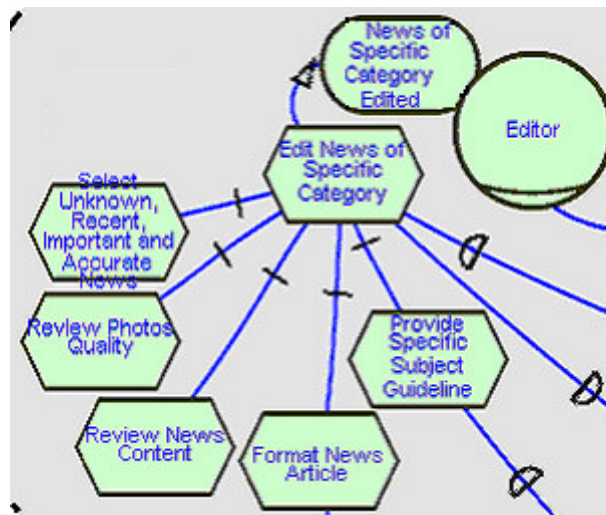


Figure 8.2 – Means-end analysis for Editor

Analogously, the Chief Editor actor expects to have the *Newspaper Edited and Published According to Guideline* and, to achieve this goal, it has the alternative of performing the *Edit and Publish Newspaper According to Guideline* task. This task is decomposed into *Request Newspaper Publishing* and *Edit Newspaper* tasks (see the refined model in Figure

8.3). The *edit newspaper* task is further decomposed into three subtasks: *Review Articles Content*, *Decompose Guideline by Category* and *Format Newspaper Pages*.

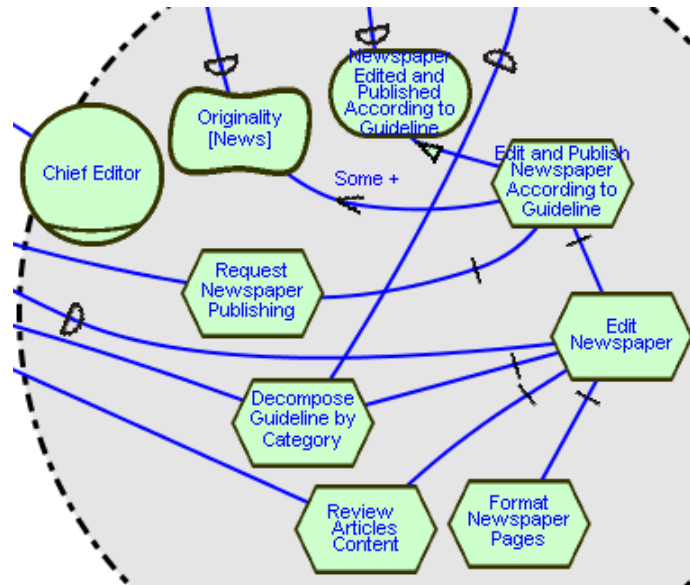


Figure 8.3 – Means-end analysis for Chief Editor

The Webmaster actor is in charge of having the *News Published* and to accomplish this goal it has two alternatives which are achieving the *Publish News Searched* or *Publish Newspaper* tasks. This first alternative is decomposed into *Get Keyword*, *Search byKeyword* and *Release Searched News* sub-tasks. This second alternative is decomposed into *Get Newspaper*, *Release Newspaper* and *Evaluate Newspaper Suitability* sub-tasks (see the refined model in Figure 8.4).

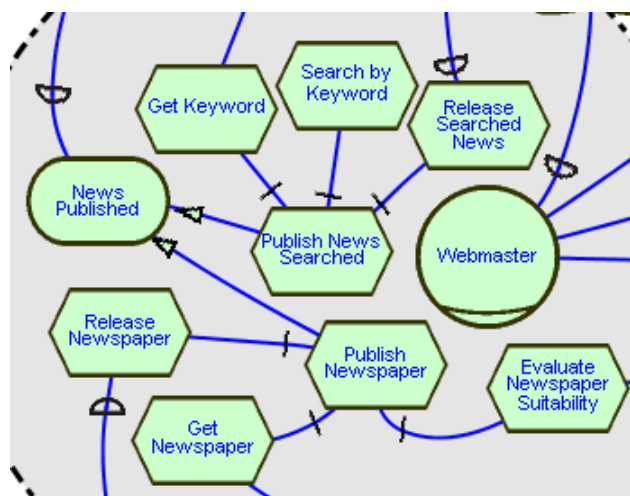


Figure 8.4 – Means-end analysis for Webmaster

The Photographer-Reporter actor is in charge of having the News Article of Specific Subject Produced and to reach this goal it has one alternative which is performing the *Contact News Agencies* task. This task is decomposed into *Link to News Agencies* and *Produce News Article* sub-tasks (see the refined model in Figure 8.5).

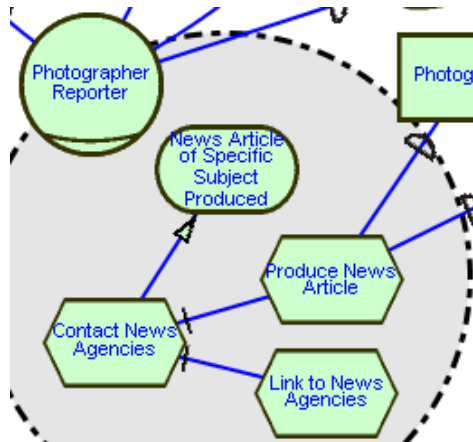


Figure 8.5 – Means-end analysis for Photographer-Reporter

8.2.1 Architectural diagram

Having concluded the means-end analysis of the actors Editor, Webmaster, Chief Editor and Photographer-Reporter, we can now move on to identify the properties that characterize that MAS according to the agent-oriented modeling diagrams (Chapter 5).

The architectural diagram (Figure 5.4) is useful to model the system's structure in terms of agent roles which depends on each other to accomplish their goals. They exchange messages according to a communication protocol through connectors attached to the agent roles by ports. Moreover, this diagram explicitly models the agent role's goals and the plans outlined to achieve these goals. The heuristics presented at Section 7.1.1 can be of some assistance to describe the Editor, Webmaster, Chief Editor and Photographer-Reporter actors according to this diagram. For example, the *News of Specific Category Edited* goal present in the means-end analysis of the Editor actor becomes a «Goal» associated to the Editor «AgentRole» (shaded area of Figure 8.6). The *Edit News of Specific Category* task becomes a «MacroPlan» associated to both the Editor «AgentRole» and News of Specific Category Edited «Goal». Each of the *Format News Article*, *Select Unknown*, *Recent*, *Important and Accurate News*, *Review Photos Quality*, *Review News Content* and *Provide*

Specific Subject Guideline tasks becomes a «ComplexAction» in the *Edit News of Specific Category* «MacroPlan».

Each dependum of the Editor actor's dependencies becomes a «Dependum». For example, the dependum of the *Provide News Article of Specific Subject* task dependency, between Editor «AgentRole» and Photographer-Reporter «AgentRole», becomes a *Provide News Article of Specific Subject* «Dependum», typed as a task (shaded area in Figure 8.6). The depender and dependee roles of the *Provide News Article of Specific Subject* dependency become the «Depender» dependency from Editor «AgentRole» and the «Dependee» realization from Photographer-Reporter «AgentRole», respectively. The *Provide News Article of Specific Subject* task dependency itself becomes a «Connector» between Editor «AgentRole» and Photographer-Reporter «AgentRole». Ports are added to these «AgentRole» classes to enable the link through the «Connector». The same guidelines are applied to the other actors present in the i* model. As a result several classes are incorporated in the MAS architectural diagram depicted in Figure 8.6.

The shaded area corresponds to the interaction between the Photographer-Reporter and Editor AgentRoles to achieve the *Produce News Article of Specific Subject*. The Editor AgentRole intends to achieve *News of Specific Category Edited* goal by means of the *Edit News of Specific Category* macroPlan. However, to edit the news of specific category the Editor AgentRole has to request the Photographer-Reporter AgentRole to perform the *Produce News Article of Specific Subject* service. The Photographer-Reporter performs the requested service because it does not conflict with the achievement of the *News Article Produced* goal. Hence, both the requested service and the goal achievement are accomplished by means of the *Contact News Agencies* macroPlan. The description of the *Publish Newspaper on Website* and *Provide News of Specific Category* services is achieved in a similar way.

In this case study we are not concerned in specifying the interaction between the e-News system and the News Agency external system, because this interaction needs the use of the Wrapper pattern [Kolp et al., 2005]. In fact, we still need to investigate what are the consequences of using a composition of social patterns in the same system, specially

regarding to the separation of crosscutting concerns. This question can only be answered by performing an empirical study.

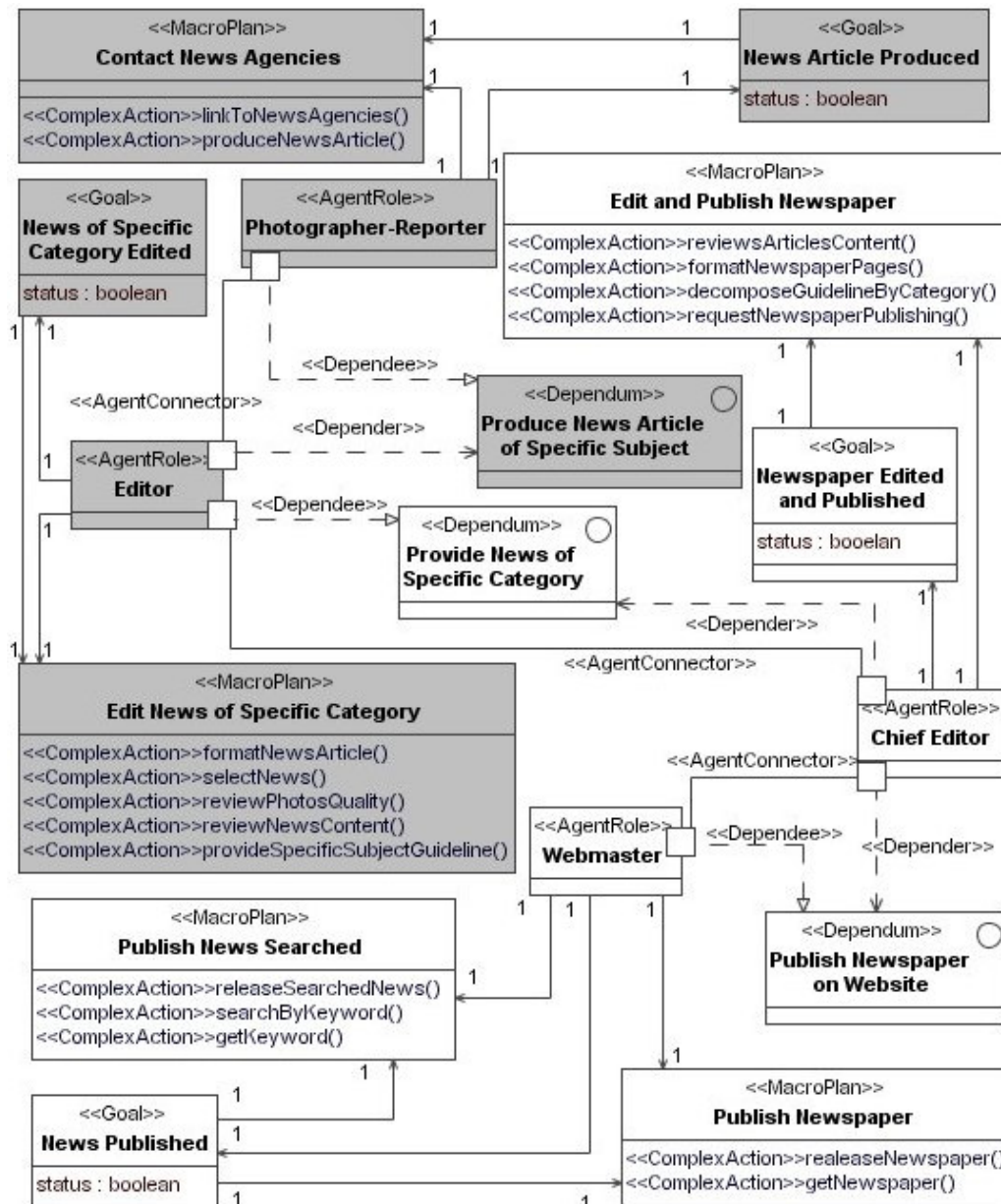


Figure 8.6 – Architectural Diagram for e-News

8.2.2 Communication diagram

The communication diagram (Figure 5.5) is useful to model the exchange of messages between two or more agents playing specific agent roles. It also captures the communication protocol used in the agents' interactions. Figure 8.7 shows an interaction

involving instances of the Chief Editor, the Editor, the Photographer-Reporter and the Webmaster AgentRoles. The interaction specified using the communication diagram is asynchronous. Hence, the Chief Editor «AgentRole» sends a message requesting news of specific category which is going to be provided through the execution of *Provide News of Specific Category* service (Figure 8.6). Then, the Editor «AgentRole» sends a message to one (or more) Photographer-Reporter «AgentRole» requesting a news article of a specific subject. The Photographer-Reporter «AgentRole», in turn, composes a news article with news and photos of a specific subject received from the News Agency external system (not shown in this diagram). Then, the Photographer-Reporter «AgentRole» answers the Editor by sending the requested news article. The Editor answers the Chief Editor by sending one (or more) pages of the newspaper which contents news articles of a specific category. The Chief Editor then edits the Newspaper and sends it to the Webmaster requesting him to publish the Newspaper. The Webmaster, in turn, answers by informing whether or not the requested service has been performed successfully.

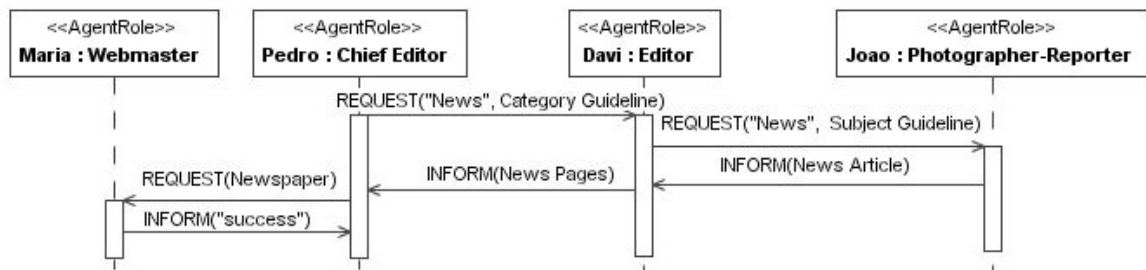


Figure 8.7 – e-News Communication Diagram

8.2.3 Environmental diagram

The environmental diagram (Figure 5.6) is useful to model the rights each agent role possesses to access resources which composes the environment where the MAS is situated. These access rights can be to create, destroy, read and write the resource. The heuristics presented in Section 7.1.1 continue to be used here to identify the properties which characterize MAS according to this diagram. Hence, all *News Pages*, *News Article*, *Subject Guideline*, *Category Guideline* resource elements related to the Editor actor (Figure 8.2) become a «Resource» associated to the Editor «AgentRole» in the environmental diagram presented in the shaded area of Figure 8.8.

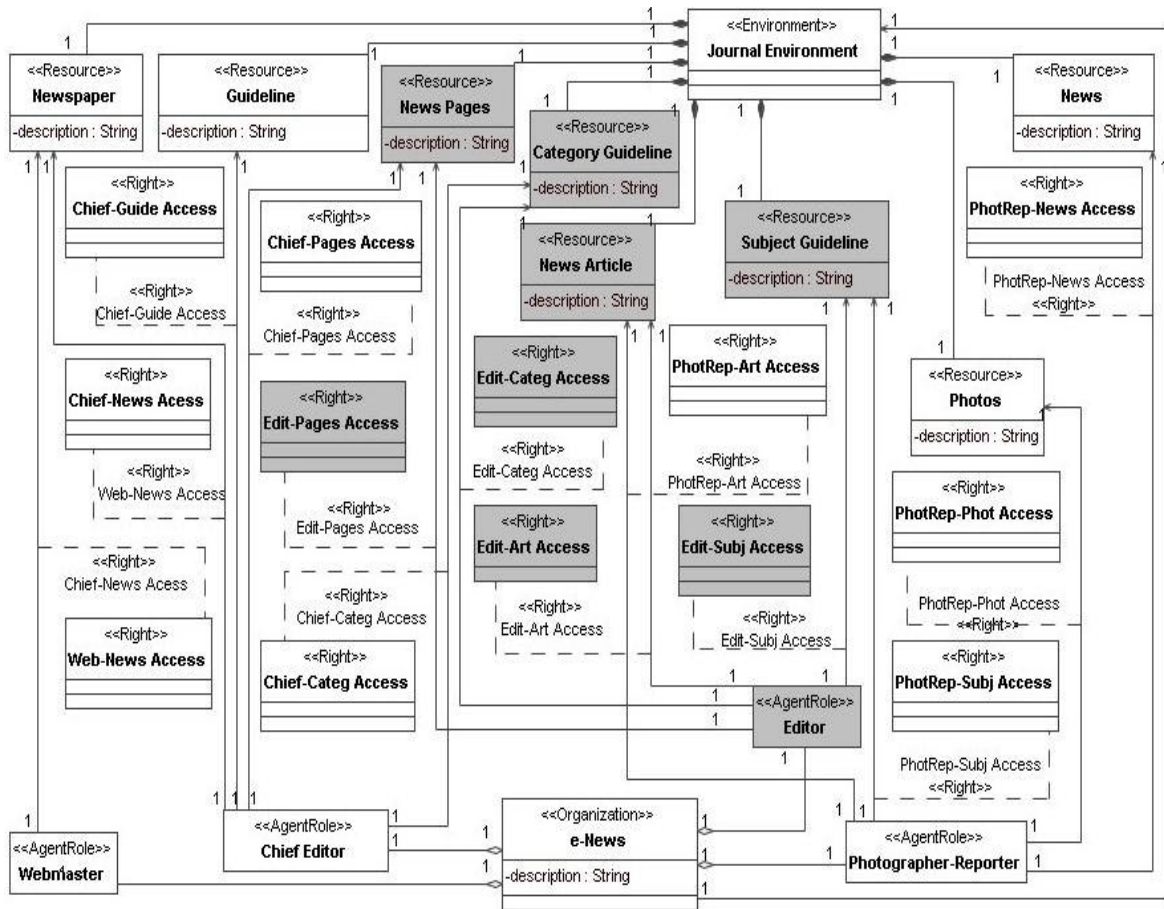


Figure 8.8 – e-News Environmental Diagram

In this diagram we have the *Editor*, *Chief Editor*, *Webmaster* and *Photographer-Reporter* AgentRoles composing the *e-News* organization which is situated in the *Journal* environment. The *Editor* «AgentRole» needs to access the *News Pages*, *News Article*, *Category Guideline* and *Subject Guideline* resource available at the *Journal* environment to achieve the *News of Specific Category Edited* goal. The *Editor* «AgentRole» can read, write, create or destroy the *News Pages* and *Subject Guideline* resources, according to its *Edit-Pages Access* and *Edit-Subj Access* rights, respectively (read *Editor-News Pages Access* right and *Editor- Subject Guideline Access* right). The *Editor* «AgentRole» can only read the *News Article* and *Category Guideline* resources, according to its *Edit-Art Access* and *Edit-Categ Access* rights, respectively (read *Editor-News Article Access* right and *Editor-Category Guideline Access* right). The description of the agent roles' rights to access resources is achieved in a similar way.

8.2.4 Intentional diagram

The intentional diagram is defined in terms of agent roles, agents, beliefs, goals, plans, intentions, norms and ontology (Figure 5.7). The heuristics presented in Section 7.1.1 are used here to identify some properties which characterize that MAS according to this diagram. Hence, the agent Davi, which plays the Editor agent role in Figure 8.1, becomes an «Agent» in the intentional diagram (shaded area of Figure 8.9). The relationship between the Davi «Agent» and the Editor «AgentRole» creates an intention to *Achieve the Editor Goal*. To accomplish this, Davi «Agent» has to execute the *Perform Editor Plan* «MicroPlan». The condition to perform the *Edit News of Specific Category* task (Figure 8.2) is that the News Article has been received. Hence, the Davi «Agent» has the *News Article Received* «Belief» which is the condition to execute the *Perform Editor Plan* «MicroPlan». The description of the intentional elements associated to the other agent roles is achieved in a similar way.

The intentional diagram is useful to capture the relationship between agents and the agent roles they play. When an agent plays a certain agent role, it has the intention to achieve the agent role's goals. This information is explicitly captured through a relationship between the agent (e.g., Pedro) and the goal (e.g., Newspaper edited and Published) it intends to achieve when playing a certain agent role (e.g., Chief Editor). Moreover, this diagram models the actual plans (e.g., Perform Chief Editor Plan) the agents will perform to accomplish the agent roles' goals (e.g., Newspaper edited and Published). Also, the intentional diagram shows the beliefs (e.g., News Pages Received) an agent (e.g., Pedro) has and which works as preconditions to perform certain plans (e.g., Perform Chief Editor Plan).

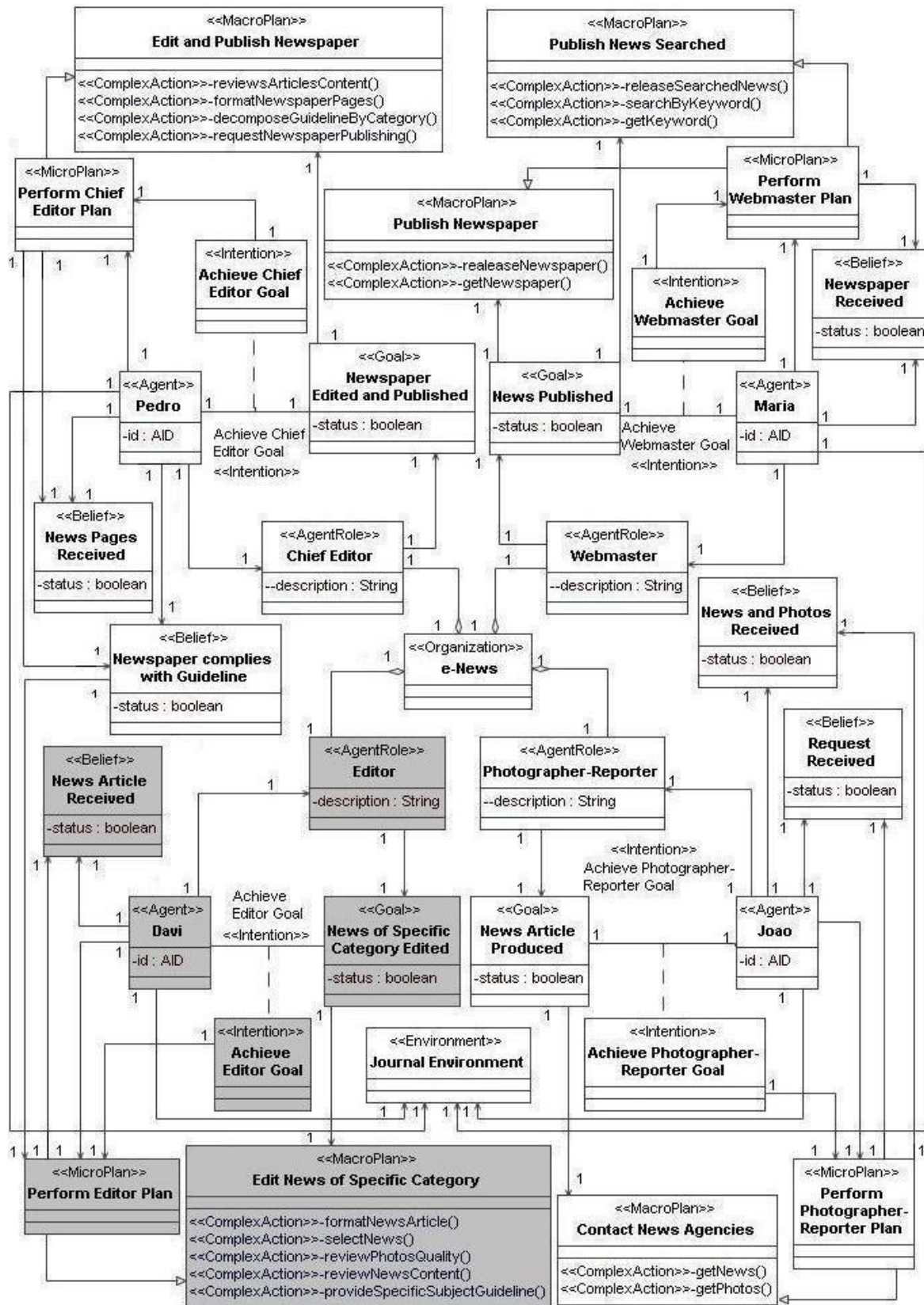


Figure 8.9 – e-News Intentional Diagram

8.2.5 Rational Diagram

The rational diagram (Figure 5.8) is defined in terms of softgoals, the contribution of plans to satisfy softgoals, goals, plans to achieve those goals and agent roles. The heuristics presented in Section 7.1.1 are used here to identify the remainder properties which characterize that MAS according to this diagram. Hence, the Originality [News] softgoal in Figure 8.3, becomes a «Softgoal» in the rational diagram (shaded area of Figure 8.10).

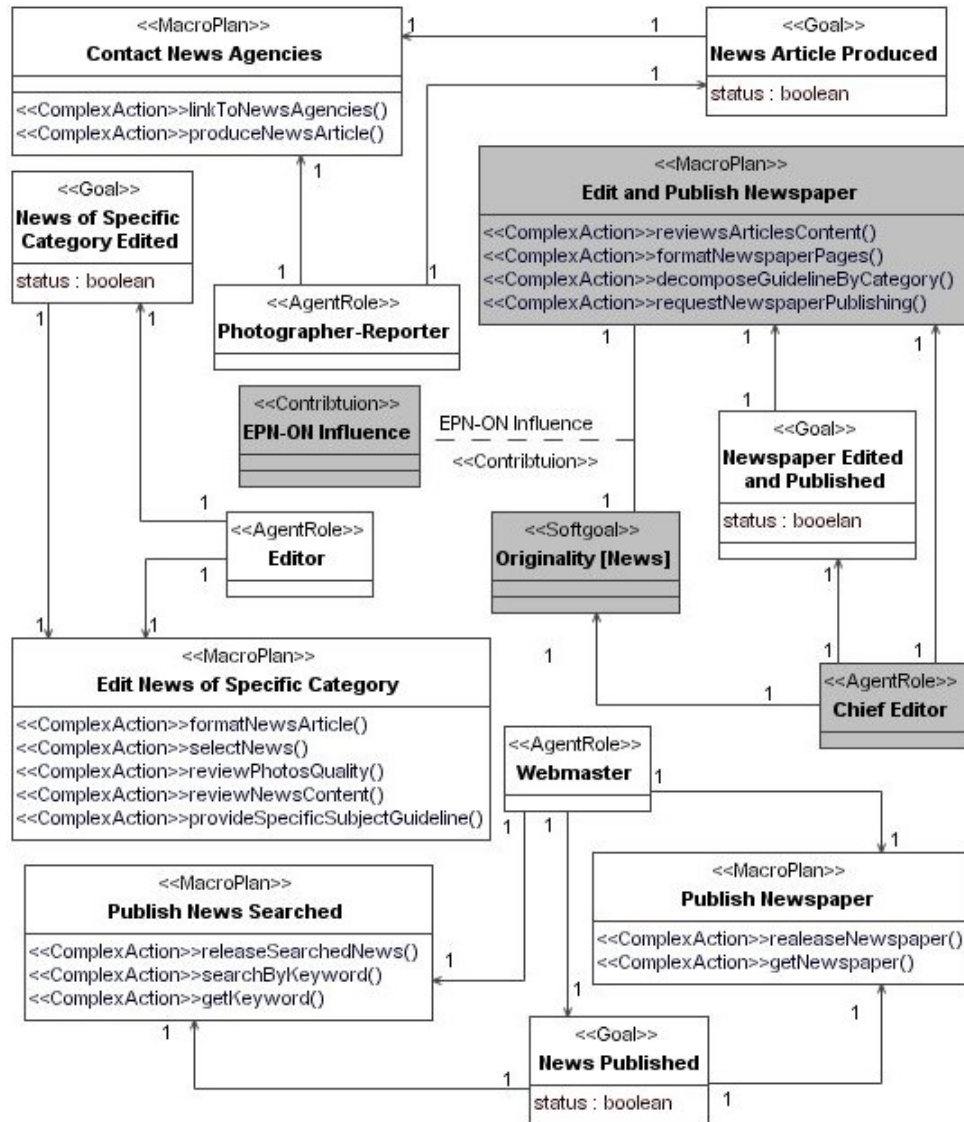


Figure 8.10 – e-News Rational Diagram

The relationship between Originality [News] «Softgoal» and Edit and Publish Newspaper «MacroPlan» creates an *EPN-ON Influence* «Contribution», which captures the positive

influence of the Edit and Publish Newspaper «MacroPlan» in the satisfaction of the Originality [News] «Softgoal». This influence analysis helps the agent in choosing the proper plan among alternatives to achieve a given goal. In the example, the Edit and Publish Newspaper «MacroPlan» is the proper choice to satisfy the Originality [News] «Softgoal» and the unique choice to satisfy the Newspaper Edited and Published «Goal».

8.2.6 Plan Diagram

The plan diagram (Figure 5.9) is defined in terms of actions which compose the plans and the sequence of these actions. The diagram depicted in Figure 8.11 details the Edit and Publish Newspaper «MacroPlan». It shows the sequence of complexActions to perform that macroPlan.

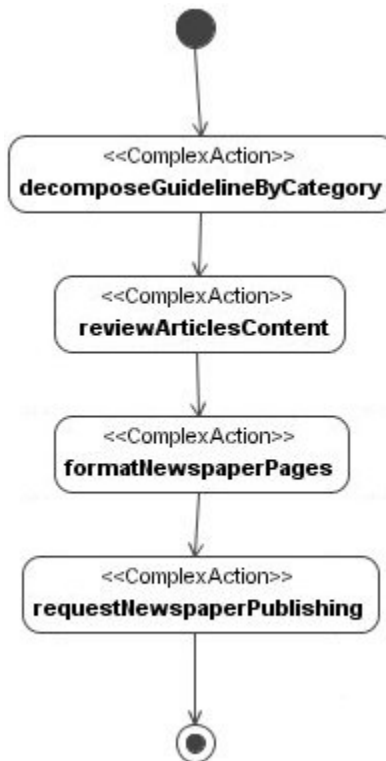


Figure 8.11 – Edit and Publish Newspaper Plan Diagram

Firstly, the Chief Editor begins by executing the `decomposeGuidelineByCategory` «ComplexAction». This complexAction decomposes and distributes the guideline to each Editor. When the Editor finishes the edition of a news article and sends it, the Chief Editor starts the `reviewArticlesContent` «ComplexAction». After that, he can perform the

formatNewspaperPages «ComplexAction». Finally, the Chief Editor performs the requestNewspaperPublishing «ComplexAction» and finishes the macroPlan execution. The remaining macroPlans and microPlans of each agent role and agent, respectively, in the system can be described using this diagram to specify in detail their behaviors.

8.3 Social Patterns Selection and Application

The last step of the process is to select and apply social patterns to refine the architectural design of the e-News system (Figure 8.6). One of the key challenges is to choose the proper social pattern to be applied to a system architectural design. One can, for example, analyze the templates (e.g. Table 6.1) that describes several features of each social pattern to address a specific requirement. For example, the e-News System has the Availability requirement (see the Availability [System] softgoal in Figure 8.1 and the requirements model in the Appendices), i.e., the system has to ensure easier recovery of the system if some agent in the system stops running. Analysing the template of several patterns, we have concluded that the most suitable pattern to address the Availability requirement is the Matchmaker, since it enables the search for another agent to replace the one that has stopped. However, in other situations, one can conclude that it would be better to use a combination of patterns to address a requirement.

To apply a social pattern, we need to weave the Matchmaker pattern with the architectural design of the e-News system. To achieve this we need to instantiate each model role present in the structure of the pattern (e.g., |Client, |Provider). For example, the following bindings represent the instantiations of the model roles present in the Matchmaker SAPS (Figure 6.1) for the e-News system:

1. Bind |Client to *Editor*
2. Bind |Provider to *Photographer-Reporter*
3. Bind |Perform Service to *Produce News Article of Specific Subject*
4. Bind |Provider.|plan.|action to *Photographer-Reporter.ContactNewsAgencies.getGuideline*
5. Bind |Client.|plan.|action to *Editor.EditNewsSpecificCategory.provideSpecificSubjectGuideline*

6. Bind `|Provider.|plan.|action` to `Photographer-Reporter.ContactNewsAgencies.produceNewsArticle`
7. Bind `|Locate Provider` to `Locate Photographer-Reporter`
8. Bind `|Matchmaker` to `Directory Facilitator`

Observe that the Matchmaker role has been instantiated to Directory Facilitator because we have chosen JADE as the target agent implementation environment.

By instantiating the model roles of the pattern (Figure 6.1) for the e-News model elements (Figure 8.6), we obtain the pattern applied to the problem (Figure 8.12). Thus, we applied the structure of the pattern to the architectural diagram of the e-News System through model roles instantiation (the bindings shown above), which can be easily automated by a CASE tool. Note that the Matchmaker pattern is refining the Editor and Photographer-Reporter AgentRoles by adding the pattern-specific concerns (ports, connectors, dependums, goals, plans). Thus, an agent playing the Editor AgentRole can search for a Photographer-Reporter at run time, as well as an agent playing the Photographer-Reporter AgentRole can publish its services in the DF's Yellow Pages.

The Matchmaker pattern ensures the decoupling among agents, because if a Photographer-Reporter contacted by an Editor stops running, for example, the Editor may replace that Photographer-Reporter by requesting to the DF to locate another one in his yellow pages. Observe in Figure 8.12 that some of the «ComplexAction» stereotypes have been omitted to save space.

It is important to highlight that the application of social patterns causes an impact in the internal design of the system agents because they are going to incorporate the ports, goals and plans of the pattern participants.

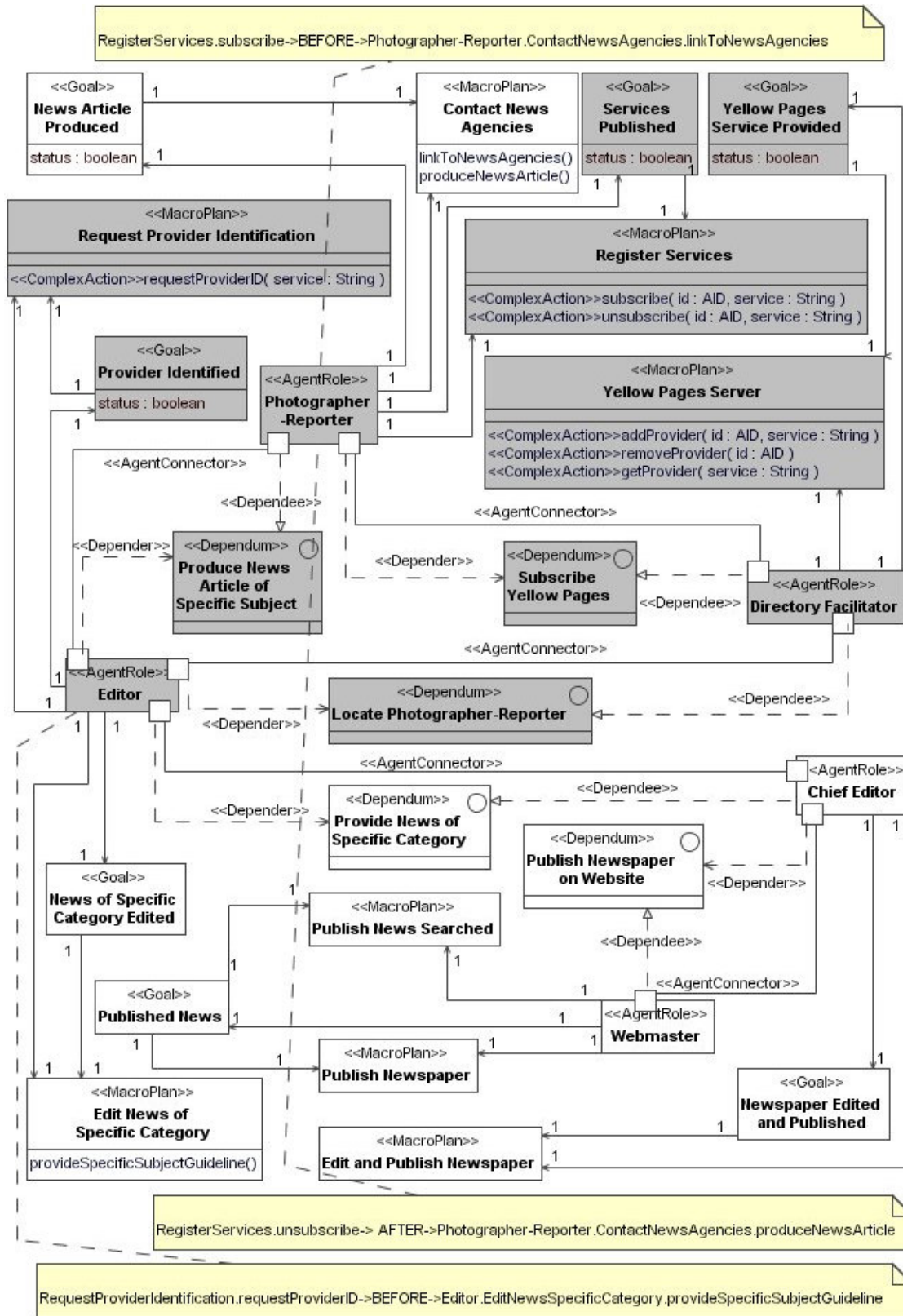


Figure 8.12 – MAS Architectural Diagram weaved with the SAPS

By instantiating the model roles present in the pattern of communications (Figure 6.2) for the e-News model elements (Figure 8.6), we specify the e-News participants in a communication pattern (Figure 8.13). The communication diagram shown in the Figure 8.13 depicts the main scenario of the communication pattern for the participants involved in the Matchmaker pattern: Joao sends a REQUEST message to subscribe its data (by sending its identification, typed by AID and its service description, typed by String) in the DF's Yellow Pages. The DF sends an INFORM message which contains the request result (e.g., "success", for the primary scenario). The Davi sends a REQUEST message to get the identification of some service provider (by sending the description of a service, typed by String). Finally, the DF sends an INFORM message which contains the search result (e.g. the provider identification, for the primary scenario).

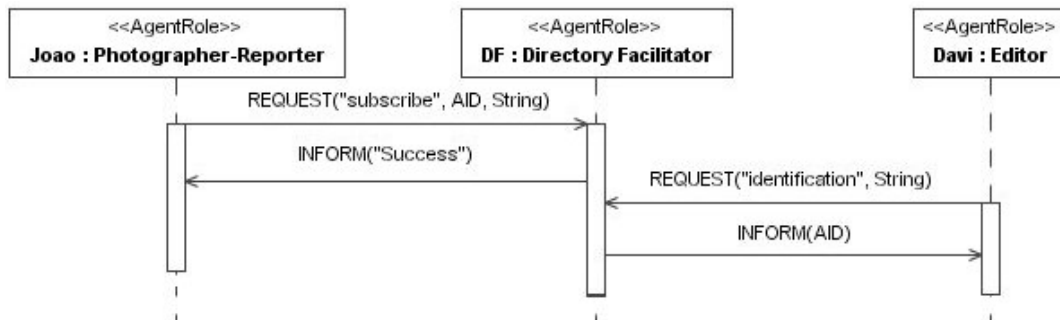


Figure 8.13 – MAS Architectural Diagram weaved with the CAPS

8.3.1 The Partial AspectJ Code for the e-News System

In this section we show how the Matchmaker Pattern introduced in Figure 6.1 can be coded using the AspectJ environment. To achieve this, we use the mapping guidelines presented in section 6.2. According to these guidelines, the MacroPlans Register Services and Request Provider Identification become aspects in AspectJ. The ComplexActions to subscribe, unsubscribe and requestProviderID become advices in its corresponding aspects. The AgentRoles IProvider and IClient are the agents affected by the aspects, i.e., the Photographer-Reporter and Editor agents. The notes become pointcuts and the composition rules in their corresponding aspects.

For example, in Figure 8.14 we present the RegisterServices aspect weaved with the PhotographerReporter agent in the e-News System.

```

public aspect RegisterServices {
    pointcut subscribe() : call (void Photographer-Reporter.linkToNewsAgencies());
    pointcut unsubscribe() : call (void Photographer-Reporter.produceNewsArticle ());
    before() : subscribe() {
        Photographer-Reporter genericAgent = (Photographer-Reporter)
thisJoinPoint.getThis();
    }
    //Register the PhotographerReporter service in the yellow pages...
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(genericAgent.getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setType(genericAgent.getServiceType());
    sd.setName(genericAgent.getServiceName());
    dfd.addServices(sd);
    try {
        DFService.register(genericAgent, dfd);
    }
    after() : unsubscribe() {
        Photographer-Reporter genericAgent = (Photographer-Reporter)
thisJoinPoint.getThis();
    }
    //Unregister from the yellow pages...
    try {
        DFService.deregister(genericAgent);
    }
}

```

Figure 8.14 – AspectJ Code for RegisterServices Aspect

Figure 8.15 presents the RequestProviderIdentification aspect weaved with the Editor agent in the e-News System.

```

public aspect RequestProviderIdentification {
    pointcut requestProviderID() : call (void
Editor.provSpecSubjGuide(..));
    before() : requestProviderID() {
        Editor genericAgent = (Editor) thisJoinPoint.getArgs()[0];
    }
    // Update the list of PhotographerReporter agents...
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType(genericAgent.getServiceType());
    template.addServices(sd);
    try {
        DFAgentDescription[] result =
DFService.search(genericAgent, template);
        for (int i = 0; i < result.length; ++i) {
            genericAgent.getProviders()[i] = result[i].getName();
        }
    }
}

```

Figure 8.15 – AspectJ Code for RequestProviderIdentification Aspect

8.3.2 The Partial JADE Code for the e-News System

In this section we show how the agents involved in the Matchmaker pattern can be coded using an integration of the AspectJ and JADE environments. According to the proposed guidelines, the agentRoles Photographer-Reporter and Editor become agents, the macroPlans become behaviors and the complexAction become methods (see Figure 8.16 and Figure 8.17).

```

public class Photographer-Reporter extends Agent {
    protected void setup() { ...}
}

```

```

public void linkToNewsAgencies () { ...}
public void produceNewsArticle () { ...}
private class ContactNewsAgencies extends Behaviour {
    public void action() { ...}
}
}

```

Figure 8.16 – JADE Code for Photographer-Reporter Agent

```

public class Editor extends Agent {
    protected void setup() { ...}
    public void provSpecSubjGuide () { ...}
    private class EditNewsSpecificCategory extends Behaviour {
        public void action() { ...}
    }
}

```

Figure 8.17 – JADE Code for Editor Agent

8.4 Summary

In this chapter we have presented the practical application of our proposal to use advanced separation of concerns when using social patterns in the development of multi-agent systems. The detailed specification of a newspaper content management system was presented, as well as a partial code in JADE and AspectJ was generated.

It is expected that the use of aspect orientation allow us to improve the locality and modularization of both agent and pattern concerns. Also, several other benefits can be achieved - improved software reusability, maintainability, understandability and evolvability. In fact, we intend to evaluate the promised benefits and possible deficiencies of using our approach by using techniques provided by the Experimental Software Engineering research area [Basili et al., 1986]. Another case study have been developed to specify, according to our approach, the well-known Conference Management System [Zambonelli et al., 2003] [Garcia, 2004]. This case study is described in the Appendices. In the next chapter, we present our conclusions and point out to future works.

9 Conclusions

This chapter presents the current results of our approach, the lessons learned during this work, as well as it points out directions to future works to improve our proposal.

9.1 Final Considerations

With the ever-increasing use of agent technology in commercial and industrial applications, we highlight the need for suitable techniques, notations and tools to support the agent-oriented software development. However, there is not much consensus yet in the literature regarding the properties required to specify MAS.

This research is a step towards a definition of the concepts and relationships to specify MAS, which were used to define an agency metamodel (by extending the UML metamodel). Several diagrams have been defined according to this agency metamodel to capture six views of MAS design. In particular, the architectural and communication diagrams have been used to develop the APS (Agent Pattern Specifications) technique, which promotes separation of social patterns concerns in MAS detailed design. Moreover, a process to guide the use of both the agent-oriented modeling notation and the social patterns described using the APS technique has been defined and described in SPEM.

Some difficulties have been faced during the development of this research. In fact, since there is not much consensus yet in the literature regarding the properties required to specify MAS, the available implementation environments do not provide all the constructs needed to fully implement a MAS. Hence, it makes it difficult performing a practical validation of any approach in the agent-oriented community, because lots of time and effort are spent to develop all the basic implementation infrastructure. Besides these difficulties, we have shown that a system specified according to our approach can be implemented using an integration of JADE and AspectJ.

9.2 Comparative Analysis

Firstly, we compare our approach with the proposal of Kolp et al. (2005) for specifying MAS design (Section 4.1). Our agent-oriented modeling notation comprises six modeling diagrams which can capture as much MAS design information as possible. For example, we have the architectural diagram which models the system structure in terms of agent roles which depends on each other to accomplish their goals. They exchange messages according to a communication protocol through connectors attached to the agent roles by ports.

Moreover, this diagram explicitly models the agent role's goals and the plans outlined to achieve these goals. The information related to agent roles is not captured by any of Kolp's diagrams. Goals are captured by Kolp's social diagram and plan is captured by Kolp's structural diagram. However, the structural diagram is dependent on JACK constructs, making it difficult to reuse the MAS design specification in other software projects.

The communication diagram models the exchange of messages between two or more agents playing specific agent roles. It also captures the communication protocol used in the agents' interactions. This information is captured by Kolp's communicational diagram. However, they do not make the distinction between agent and agent role in the design of a MAS neither use the FIPA performatives explicitly.

The environmental diagram models the rights each agent role possesses to access resources which composes the environment where the MAS is situated. These access rights can be to create, destroy, read and write the resource. This information could not be captured by Kolp's proposal.

The intentional diagrams capture the relationship between agents and the agent roles they play. When an agent plays a certain agent role, it has the intention to achieve the agent role's goals. This information is explicitly captured through a relationship between the agent and the goal it intends to achieve when playing a certain agent role. Moreover, this diagram models the actual plans the agents will perform to accomplish the agent roles' goals. Also, the intentional diagram shows the agent's beliefs which work as preconditions to perform certain plans.

The norms and the ontology the MAS organization must comply to can be also captured in this diagram. The information related to agents, beliefs and plans is captured by Kolp's structural diagram. However, information such as norms, ontology, intention, the relationship between agent and agent roles cannot be captured by Kolp's proposal.

In the rational diagram we capture the influence of plans in the satisfaction of softgoals. This influence analysis helps the agent in choosing the proper plan among alternatives to achieve a given goal. This kind of analysis is provided by the i* framework and has been

preserved by our notation. On the other hand, the proposal of Kolp et al. (2005) does not capture this information.

Finally, the plan diagram models the sequence of actions to be performed into a plan. This kind of information is captured by the Kolp's dynamic diagram.

Secondly, we compare our approach with Kolp et al. (2005) proposal for describing social patterns (Section 4.1), especially regarding the separation of patterns concerns. We proposed a technique to describe social patterns, called APS, which uses a subset of GoF's and two diagrams to capture structure and behavior patterns. The main template attributes are Name, Intent, Applicability, Motivation example, Problem, Solution and Participants. Kolp's proposal captures through the intentional dimension only the description of the pattern's participants. This dimension is concerned with the identification of *services* provided by agents which participates of the pattern. In our approach, the SAPS specializes the agency metamodel using model roles. It defines subtypes of agency metaclasses describing MAS architectural diagram elements which are going to be instantiated with design elements of a given application when the pattern is applied to it. In particular, we propose an aspect-oriented description of patterns where we assume that each participant of the pattern will be encapsulated by an aspect. We attach notes to each agent role present in the SAPS to describe the agent roles' points affected by the aspects encapsulating the pattern's concerns. The Communication Agent Pattern Specification also specializes the agency metamodel using model roles. It defines subtypes of agency metaclasses describing MAS communication diagram elements and therefore, describes a pattern of communications. The approach proposed by Kolp et al. (2005) does not take into consideration the separation of social patterns concerns in MAS design neither proposes a systematic way to apply these patterns. On the other hand, in this thesis we describe social patterns using the dynamic diagram while we are still investigating how the plan diagram can be used to complement the APS technique. The information captured by Kolp's social diagrams is all captured by the SAPS. However, the information related to belief in Kolp's structural diagram of social patterns is not captured yet by the APS technique. We are also investigating the use of the intentional diagram to describe social patterns according to the APS technique. To demonstrate the feasibility of the APS technique we provide hints about

the implementation of MAS using advanced separation of social patterns concerns. In fact, some guidelines to map our technique to describe social patterns in constructs of JADE and AspectJ are provided.

Finally, we compare our detailed design process with the proposal of Wautelet et al. (2005) to perform MAS detailed design in Tropos (Section 4.3). Our process starts by bridging the gap between the use of the i^* notation in the earlier phases of Tropos and the use of an UML-based notation in the detailed design phase. We propose heuristics to derive a detailed specification of MAS design using some UML-based diagrams from a description of MAS architectural design using i^* . Thus, our process presents some activities for guiding the use of the agent-oriented modeling diagrams. Moreover, the process provides activities to help the selection of social patterns to a specific MAS application and to systematically guide the application of the chosen patterns. The process described in Wautelet et al. (2005) uses the proposal of Kolp et al. (2005) both to describe social patterns and specify MAS design. That process does not provide an explicit and systematic way for selecting and applying the social patterns during MAS design. Therefore, the patterns are applied in an ad hoc way and their choice is not properly justified.

9.3 Contributions

This thesis aims to address current issues related to the lack of proper techniques and notation to both specify MAS design and describe social patterns using advanced separation of concerns. The contributions are: (i) an agency metamodel (by extending the UML metamodel); (ii) a notation for specifying MAS designs which comprises six UML-based diagrams; (iii) a standard technique, called APS, to describe the social patterns concerns separately from MAS functional modules (i.e. agent roles); (iv) a process to guide the design of MAS according to our notation as well as the choice and application of social patterns to a system under development; and (v) guidelines to implement a system specified according to our approach using the integration of JADE and AspectJ environments.

The MAS modeling diagrams were defined according to the agency metamodel to capture MAS design. It includes six UML-based diagrams capturing several views of MAS, such as Architectural, Communication, Intentional, Environmental, Rational and Plan. Besides,

supporting the BDI model and FIPA standards, the proposed diagrams also address MAS architectural constructs (ports, connectors, interfaces and protocols) and constructs such as agent role, intention, commitment, trust, agent communication language, degree of dependency, contribution, etc. Finally, these diagrams are supported by any UML CASE tool and may be used in other frameworks/methodologies, facilitating the development and design of MAS.

To describe the structure and behavior of agent-oriented design patterns, we proposed the APS technique. This technique describes social patterns using abstractions and mechanisms provided by aspect orientation to support the separation of crosscutting concerns in MAS. In doing so, several benefits for MAS design and implementation can be achieved, such as improving concerns modularity, since social patterns concerns are localized into aspects while the application concerns are localized into agent roles. Consequently, this improves software reusability and maintainability. Since the APS technique is based on a standard modeling language (UML), the social patterns could be easily used by several agent-oriented methodologies as well as modeled by any UML CASE tool.

A process to perform detailed design of MAS using the MAS modeling notation and the social patterns described using the APS technique has been defined and described using SPEM. The process includes activities to map i* models to UML-based diagrams as well as to select and apply social patterns to introduce additional detail in MAS design. Moreover, the diagrams produced in the detailed design phase, according to our approach, can be implemented using an integration of JADE and AspectJ environments.

9.4 Future Work

The case study has helped us observing that some information associated to MAS is not yet addressed by the current mapping heuristics. For example, strategies to derive several agent features (e.g. system ontology, organizational norms, agent communication language, etc.) from i* models are still required. Furthermore, the specification of the Communication diagrams, «MicroPlan» and «BasicAction» are still performed in an *ad hoc* fashion. These issues will be addressed in future work.

Future work also includes: (i) performing more complex case studies; (ii) complementing the agency profile by defining OCL constraints [OMG, 2005b]; (iii) increasing the expressiveness of the MAS modeling diagrams, mainly the plan diagram; (iv) improving the detailed design process; (v) investigating if the intentional, environmental, rational and plan diagrams can be used to specify social patterns and complement the template used to describe the social patterns; (vi) providing a complete catalogue of social patterns described using this technique; (vii) developing a tool to support both the process and the code generation; (viii) handling conflicts which might occur when different social patterns are applied to the same MAS design; (ix) applying the approach to more case studies for validation purposes; (x) investigating the feasibility of implementing in JADEX a MAS specified according to our approach; (xi) performing an empirical evaluation of the benefits of using aspects in the implementation of social patterns. Firstly, we need to implement MAS using only JADE, then we use the integration of JADE and AspectJ. Then, we assess the degree of modularity of MAS in each case and compare the results; and, finally, (xii) investigating through empirical study, what are the consequences of using a composition of social patterns in the same system, specially regarding to the separation of crosscutting concerns.

References

- [Alexander, 1979] Alexander, C.: The Timeless Way of Building. Oxford University Press, New York, ISBN: 0195024028, 1979.
- [AOS, 2005] Agent Oriented Software Group. JACK Intelligent Agents. <http://www.agent-software.com/>, Last access in March 2005.
- [AOSD, 2003] AOSD Mailing Discussion List. Available at: <http://aosd.net>. Last access in November 2003.
- [Araújo et al., 2004] Araújo, J., Whittle, J., Kim, D.: Modeling and Composing Scenario-Based Requirements with Aspects. In: 12th IEEE International Requirements Engineering Conference (RE'04), Kyoto, Japan, September 2004.
- [Aridor and Lange, 1998] Aridor, Y. and Lange, D. B.: Agent design patterns: elements of agent application design. In: 2nd International Conference on Autonomous Agents, Minneapolis, Minnesota, United States. ISBN:0-89791-983-1, pp.: 108 – 115, 1998.
- [Aksit et al., 2001] Aksit, M., Tekinerdogan, B. and Bergmans, L.: The Six Concerns for Separation of Concerns. In: Workshop on Advanced Separation of Concerns at ECOOP'01, Budapest, Hungary, June 18-22, 2001.
- [Asimov, 1968] Asimov, I.: I, Robot (a collection of short stories originally published between 1940 and 1950), Grafton Books, London, 1968.

- [Basili et al., 1986] Basili, V.; Selby, R.; Hutchen, D.: Experimentation in Software Engineering. In: IEEE Transactions on Software Engineering, 12, 7, pp.:733-743, 1986.
- [Bass et al., 1998] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Reading, MA: Addison-Wesley Longman, 1998.
- [Bastos, 2005] Bastos, L. R. D.: Integration of System Requirements and Multi-agent Software Architecture. Ph.D Thesis. Centro de Informática, Universidade Federal de Pernambuco, August 2005.
- [Bellifemine et al., 2003] Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE - A White Paper. In: Special issue on JADE of the TILAB Journal EXP, 2003.
- [Bihari et al., 1989] Bihari, T., Gopinath, P., Schwan, K.: Object-Oriented Design of Real-Time Software. In: 10th Real-Time Systems Symposium. Los Alamitos, Califórnia, pp.: 194 – 201, 1989.
- [Braubach et al., 2004] Braubach, L., Pokahr, A. and Lamersdorf, W.: Jadex: A Short Overview, In: Main Conference Net.ObjectDays 2004, AgentExpo.
- [Brazier et al., 2001] Brazier, F., Dunin-Kępicz, B., Treur, J., Verbrugge, R.: Modelling Internal Dynamic Behaviour of BDI Agents. In: Gabbay, D., Smets, Ph. (Eds.): Dynamics and Management of Reasoning Processes. Series in Defeasible Reasoning and Uncertainty Management Systems, Vol. 6. Kluwer Academic Publishers, pp. 339 – 361, 2001.
- [Cacho et al., 2006] Cacho, N., Sant'Anna, C., Figueiredo, E., Garcia, A., Batista, T., Lucena, C.: Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming. In: 5th International Conference

- on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany, 2006.
- [Camargo et al., 2003] Camargo, V., Ramos, R., Penteado, R., Masiero, P.: Projeto Baseado em Aspectos do Padrão Camada de Persistência. In: Simpósio Brasileiro de Engenharia de Software (SBES'03), Manaus, 2003.
- [Castro et al., 2002] Castro, J. Kolp, M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. In: Information Systems Journal. Volume 27. Elsevier, pp.: 365-89, 2002.
- [Castro et al., 2003] Castro, J., Silva, C., Mylopoulos, J.: Detailing Architectural Design in the Tropos Methodology. In: 15th Conference Advanced Information Systems Engineering (CAiSE'03). Klagenfurt/Velden, Austria, pp. 111 – 126, 2003.
- [Chavez et al., 2004] Chavez, C.: A Model-Driven Approach to Aspect-Oriented Design. Ph.D Thesis, Computer Science Department, PUC-Rio, April 2004.
- [Chavez et al., 2005] Chavez, C., Garcia, A., Kulesza, U., Sant'Anna, C., Lucena, C.: Taming Heterogeneous Aspects with Crosscutting Interfaces. In: 19th Brazilian Symposium on Software Engineering (SBES'05), Uberlandia, Brazil, October 2005.
- [Clarke and Walker, 2001] Clarke, S. and Walker, R. J.: Composition Patterns: An Approach to Designing Reusable Aspects. In: 23rd International Conference on Software Engineering (ICSE'01), 2001.
- [Deugo et al., 1999] Deugo, D., Oppacher, F., Kuester, J., Von Otte, I.: Patterns as a Means for Intelligent Software Engineering. In: International Conference on Artificial Intelligence, School of Computer

- Science, Carleton University, Ottawa, Ontario, Canada, pp.: 605-611, 1999.
- [Dignum and Dignum, 2001] Dignum, F. P. M. and Dignum, M. V.: Modelling Agent Societies: Coordination Frameworks and Institutions. In: Progress in Artificial Intelligence. pp.: 191-204. Springer-Verlag, 2001.
- [Dijkstra, 1976] Dijkstra, E.: A Discipline of Programming. Prentice-Hall, 1976.
- [Elrad et al., 2001] Elrad, T., Filman, R. and Bader, A.: Aspect-Oriented Programming: Introduction, In: Communications of the ACM, v.44 n.10, pp.: 29-32, October 2001.
- [FIPA, 2004] FIPA: The Foundation for Intelligent Physical Agents, Available: <http://www.fipa.org>, 2004. Last access in November 2006.
- [Ferber, 1999] Ferber, J.: Multiagent Systems: An Introduction to Distributed Artificial Intelligence. Addison Wesley, 1999
- [Ferber and Gutknecht, 2000] Ferber, J. and Gutknecht, O.: Admission of agents in groups as a normative and organizational problem. In: Workshop on Norms and Institutions in Multi-agent systems at Normative Agent, ACM press, 2000.
- [France et al., 2004] France, R. Kim, D. Ghosh, S. and Song, E.: A UML-Based Pattern Specification Technique In: IEEE Transactions on Software Engineering, vol. 30, pp. 193-2006, 2004.
- [Gamma et al., 1994] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Reading, USA, 1994. ISBN: 0201633612.

- [Garcia, 2004] Garcia, A.: From Objects to Agents: An Aspect-Oriented Approach. PhD Thesis, Computer Science Department, PUC-Rio, Rio de Janeiro, Brazil, 2004.
- [Garcia et al., 2005] Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., von Staa, A.: Modularizing Design Patterns with Aspects: A Quantitative Study. In: 4th International Conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA, 2005.
- [Georg et al., 2002] Georg, G. Ray I. and France, R.: Using Aspects to Design a Secure System. In: 8th IEEE International Conference on Engineering of Complex Computer Systems, Greenbelt, Maryland, 2002.
- [Ghezzi et al., 1991] Ghezzi, C.; Jazayeri, M. and Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall, 1991.
- [Giorgini et al., 2005] Giorgini, P., Kolp, M., Mylopoulos, J. and Castro, J.: Tropos: A Requirements-Driven Methodology for Agent-Oriented Software, Book Chapter in Agent-Oriented Methodologies. ed.: Idea Group, pp. 20-45, 2005.
- [Green et al., 1997] Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F., Evans, R.: Software Agents: A Review. Technical report. The Intelligent Agents Group. Trinity College Dublin. Broadcom Eireann Research, Ltd., 1997.
- [Guizzardi and Wagner, 2005] Guizzardi, G. and Wagner, G.: Towards Ontological Foundations for Agent Modeling Concepts using UFO. In: Lecture Notes on Artificial Intelligence (LNAI) 3508, Springer-Verlag, 2005.

- [Guizzardi, 2006] Guizzardi, R.: Agent-oriented Constructivist Knowledge Management. PhD Thesis. University of Twente. The Netherlands, 2006.
- [Hannemann and Kiczales, 2002] Hannemann, J. and Kiczales, G.: Design Pattern Implementation in Java and AspectJ. In: 17th Annual ACM conference on Object-Oriented Programming, Systems, Languages, and Applications (*OOPSLA'02*), pp. 161-173, 2002..
- [Hayden and Yang, 1999] Hayden, C. C. and Yang, Q.: Architectural Design Patterns for Multiagent Coordination. In: 3rd International Conference on Agent Systems, Seattle, USA, 1999.
- [Hursch and Lopes, 1995] Hursch, W. and Lopes, C.: Separation of Concerns. Technical Report NU CCS-95-03, Northeastern University, 1995.
- [IEEE, 2000] IEEE Computer Society: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems: IEEE Std 1471, 2000.
- [ISO 9126, 1996] ISO 9126: Information Technology - Software quality characteristics and metrics. ISO/IEC, 1996.
- [Java, 2005] Java Technology: <http://www.java.sun.com/>, Last access in March 2005.
- [Jennings, 2000] Jennings, N. R.: On Agent-Based Software Engineering. Artificial Intelligence, 117(2): 277-296, 2000.
- [Jennings, 2003] Jennings, N. R.: Applying Agent Technology. In: 5th European Agent Systems Spring School (EASSS'03). Barcelona, Spain, 2003.

- [Kendall et al., 1998] Kendall, E. A., Krishna, P. V. M., Pathak, C. V., Suresh, C. B.: Patterns of Intelligent and Mobile Agents. In: 2nd International Conference on Autonomous Agents, Minneapolis, Minnesota, United States, pp.: 92 – 99, ISBN:0-89791-983-1, 1998.
- [Kersten and Murphy, 1999] Kersten, M. and Murphy, G.: Atlas: A Case Study in Building a Web-Based Learning Environment using Aspect-oriented Programming. In OOPSLA'99, 1999.
- [Kiczales et al., 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-Oriented Programming. In: European Conference on Object-Oriented Programming. (ECOOP'97), Finland, Springer-Verlag, 1997.
- [Kiczales et al., 2001] Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J. and Griswold, W.: An Overview of AspectJ. In: J. L. Knudsen, editor: 15th European Conference on Object-Oriented Programming (ECOOP'01), volume 2072 of LNCS, pp.: 327–353, Berlin, Heidelberg, and New York, Springer Verlag, 2001.
- [Kim et al., 2003] Kim, D., France, R., Ghosh, S. and Song, E.: A Role-Based Metamodeling Approach to Specifying Design Patterns. In: 27th IEEE Annual International Computer Software and Applications Conference (COMPSAC'03), November 2003.
- [Kolp et al., 2001a] Kolp, M., Castro, J., Mylopoulos, J.: A social organization perspective on software architectures. In: 1st International Workshop From Software Requirements to Architectures (STRAW'01), Toronto, Canada, pp.: 5-12, 2001.
- [Kolp et al., 2001b] Kolp, M., Giorgini, P., Mylopoulos, J.: A goal-based organizational perspective on multi-agents architectures. In: 8th International Workshop on Intelligent Agents: Agent Theories,

- Architectures, and Languages (ATAL'01), Seattle, USA, 2001.
- [Kolp et al., 2002] Kolp, M., Giorgini, P., Mylopoulos, J.: Information Systems Development through Social Structures. In: 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ishia, Italy, 2002.
- [Kolp et al., 2005] Kolp, M., Do, T. T., Faulkner, S. and Hoang, H. T. T.: Introspecting Agent Oriented Design Patterns, In: S. K. Chang (Eds.), Advances in Software Engineering and Knowledge Engineering, vol. III, World Publishing, 2005.
- [Lieberherr and Holland, 1989] Lieberherr, K. and Holland, I.: Assuring Good Style for Object-Oriented Programs. IEEE Software, pp.: 38-48, September 1989.
- [Luck et al., 2003] Luck, M., McBurney, P., Preist, C.: Agent technology roadmap: Enabling next generation computing. AgentLink report. Available at www.agentlink.org/roadmap, 2003. Last access in December 2005.
- [Minsky and Muarata, 2004] Minsky, N and Muarata, T.: On Manageability and Robustness of Open Multi-Agent Systems. In: Carlos Lucena and Alessandro Garcia and Alexander Romanovsky and Jaelson Castro and Paulo Alencar (Eds.), Computer Security, Dependability, and Assurance, LNCS, No. 2940, Springer-Verlag, pp.: 189-206, 2004.
- [Moraitis et al., 2002] Moraitis, P., Petraki, E. and Spanoudakis, N.: Engineering JADE Agents with the Gaia Methodology. In: R. Kowalczyk, J. Muller, H. Tianfield, R. Unland (Eds.), Agent Technologies, Infrastructures, Tools and Applications for E-Services, Best (revised) papers of NODE'02 Agent-Related Workshops,

- (LNAI2592), pp.: 77-92, 2002.
- [Mouratidis et al., 2005] Mouratidis, H., Faulkner, S., Kolp, M., Giorgini, P.: A Secure Architectural Description Language for Agent Systems. In: 4th Autonomous Agents and Multi-Agent Systems (AAMAS'05), The Netherlands, 2005.
- [Muller, 1998] Muller, J. P.: The Right Agent (Architecture) to Do the Right Thing. In: Intelligent Agents V, 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98), LNCS, Vol. 1555, Springer-Verlag, pp.: 211 - 225, 1998.
- [Mylopoulos et al., 2001] Mylopoulos, J., Kolp, M., Castro, J.: UML for agent-oriented software development: The Tropos proposal. In: 4th Unified Modeling Language (UML'01), Toronto, Canada, 2001.
- [Noda and Kishi, 2001] Noda, N. and Kishi, T.: Implementing Design Patterns Using Advanced Separation of Concerns. In: Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA'01), Tampa Bay, FL, October 2001.
- [Odell et al., 2000] Odell, J., Parunak, H. V. D, Bauer, B.: Extending UML for agents. In: 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems at the 17th National Conference on Artificial Intelligence, Austin, USA. iCue Publishing, pp.: 3–17, 2000.
- [OMG, 1997] Object Management Group: Object Constraint Language Specification, version 1.1, OMG document ad970808, 1997.
- [OMG, 2005a] Object Management Group: Agent Platform Special Interest Group, Available at <http://www.omg.org/agent/>, 2005. Last access in November 2005.

- [OMG, 2005b] Object Management Group: Unified Modeling Language (UML): Superstructure. Version 2.0, Available: www.omg.org/docs/formal/05-07-04.pdf, 2005. Last access in November 2006.
- [OMG, 2006] Object Management Group: Software Process Engineering Metamodel (SPEM) Specification. Version 1.1, Available: www.omg.org/docs/formal/05-01-06.pdf, 2006. Last access in December 2006.
- [Penserini et al., 2006] Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From Stakeholder Intentions to Software Agent Implementations. In: 18th Conference on Advanced Information Systems Engineering (CAiSE-06), Springer Verlag, LNCS-4001, Luxembourg, June 2006.
- [Pressman, 2003] Pressman, R. S.: Software Engineering – A Practitioner’s Approach, McGraw Hill, 5th Edition, 2003.
- [Quenum and Akinine, 2005] Quenum, J. G., Akinine, S.: A Dynamic Joint Protocols Selection Method to Perform Collaborative Tasks. In: 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS’05), pp. 11-20, 2005.
- [Quenum et al., 2003] Quenum, J. G., Slodzian, A., Akinine, S.: Automatic Derivation of Agent Interaction Model from Generic Interaction Protocols. In: Agent-Oriented Software Engineering (AOSE’03), pp. 138-152, 2003.
- [Rao and Georgeff, 1995] Rao, A. S. and Georgeff, M. P.: BDI agents: from theory to practice, Technical Note 56, Australian Artificial Intelligence Institute, 1995.

- [Sant'Anna et al., 2004] Sant'Anna, C., Garcia, A., Kulesza, U., Lucena, C. and von Staa, A.: Design Patterns as Aspects: A Quantitative Assessment. In: Journal of the Brazilian Computer Society, SBC, 2004.
- [Selic and Rumbaugh, 1998] Selic, B. and Rumbaugh, J.: Using UML for Modeling Complex Real - Time Systems. Rational Whitepaper, www.rational.com, 1998. Last access in December 2003.
- [Shaw and Garlan, 1993] Shaw, M. and Garlan, D.: An Introduction to Software Architecture. In: Ambriola, V. and Tortora, G. (Eds.), Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific Publishing Company, New Jersey, 1993.
- [Silva, C. et al., 2003] Silva, C., Pinto, R. C. C., Castro, J. B. F., Tedesco, P. C. A. R.: Requirements for Multi-Agent Systems. In: VI Workshop em Engenharia de Requisitos (WER03), Piracicaba, Brasil, 2003.
- [Silva, V. et al., 2003] Silva, V., Garcia, A., Brandão, A., Chavez, C., Lucena, C., Alencar, P.: Taming Agents and Objects in Software Engineering. In: Garcia, A. et al. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems, LNCS, Vol. 2603. Springer-Verlag, pp. 1 – 25, 2003.
- [Silva, C. et al., 2004] Silva, C., Tedesco, P. C, Castro, J. B. F., Pinto, R. C. C.: Comparing Agent-Oriented Methodologies Using a NFR Approach. In: Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'04), 2004.
- [Silva and Lucena, 2004] Silva, V. and Lucena, C.: From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language, In: Sycara, K., Wooldridge, M. (Eds.), Journal of Autonomous Agents and Multi-Agent Systems, Kluwer

- Academic Publishers, ISSN 1387-2532, volume 9, issue 1-2, pp. 145-189, 2004.
- [Silva, I. 2005] Silva, I. G. L.: Design and Implementation of Multi-agent Systems: The Tropos Case. (In portuguese: Projeto e Implementação de Sistemas Multi-Agentes: O Caso Tropos). Master Thesis. Centro de Informática, Universidade Federal de Pernambuco, September 2005.
- [Silva, V. et al., 2005] Silva, V. T., Noya, R. C., Lucena, C. J. P.: Using the UML 2.0 activity diagram to model agent plans and actions. In 4th International Joint Conference on Autonomous Agents and Multiagent Systems, The Netherlands. pp.: 594 – 600, 2005.
- [Silva, C. et al., 2005] Silva, C., Castro, J., Tedesco, P., Silva, I.: Describing Agent-Oriented Design Patterns in Tropos. In: Brazilian Symposium of Software Engineering (SBES'05), Brazil, pp. 10-25, 2005.
- [Silva, C. et al., 2006a] Silva, C., Castro, J., Araújo, J., Moreira, A., Alencar, F., Ramos, R.: Separation and Modularization of Crosscutting Social Patterns in Detailed Architectural Design. In: 18th International Conference Advanced Information Systems Engineering (CAiSE'06 Forum), Tudor, Luxembourg, 2006.
- [Silva, C. et al., 2006b] Silva, C., Castro, J., Tedesco, P., Araújo, J., Moreira, A., Mylopoulos, J.: Improving the Architectural Design of Multi-Agent Systems: The Tropos Case, In: 5th Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'06) at ICSE'06, Shanghai, China, pp. 107 – 113, 2006.
- [Silva, C. et al., 2006c] Silva, C., Araújo, J., Moreira, A., Castro, J., Tedesco, P., Alencar, F., Ramos, R.: Modeling Multi-Agent Systems using UML. In: 20th Brazilian Symposium on Software Engineering

- (SBES'06), Florianópolis, Brazil, pp. 81 – 96, 2006.
- [Silva, C. et al., 2006d] Silva, C., Castro, J., Tedesco, P., Araújo, J., Moreira, A., Mylopoulos, J.: Improving Multi-Agent Architectural Design, In: SELMAS V LNCS, 2006
- [Silva, C. et al., 2006e] Silva, C., Araújo, J., Moreira, A., Castro, J., Penaforte, D., Carvalho, A.: Towards an Aspect Oriented Modeling in Multi-agent In: 3rd Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'06), Florianópolis, 2006.
- [Silva, C. et al., 2006f] Silva, C., Araújo, J., Moreira, A., Castro, J., Alencar, F., Ramos, R.: Organizational Architectural Styles Specification. In: Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06), Barcelona, 2006.
- [Silva, C. et al., 2007a] Silva, C., Araújo, J., Moreira, A., Castro, J.: Designing Social Patterns using Advanced Separation of Concerns. In: 19th International Conference Advanced Information Systems Engineering (CAiSE'07), Trondheim, Norway, 11-15 June 2007 .
- [Silva, C. et al., 2007b] Silva, C., Araújo, J., Moreira, A., Castro, J.: Towards a Standardized Description and a Systematic Use of Social Patterns. In: 10th Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS'07), Isla de Margarita, Venezuela, 7-11 May 2007.
- [Sommerville, 2004] Sommerville, I. Software Engineering – Ed.7. Addison Wesley, 2004
- [Sousa, 2004] Sousa, G.: An Use Case Driven Approach for Aspect-Oriented Software Development. (In portuguese: Uma Abordagem

- Direcionada a Casos de Uso para o Desenvolvimento de Software Orientado a Aspectos). Master Thesis. Centro de Informática, Universidade Federal de Pernambuco, September 2004.
- [Stein, 2002] Stein, D.: An Aspect-Oriented Design Model Based on AspectJ and UML, Master Thesis, University of Essen, January, 2002.
- [Sturm and Shehory, 2003] Sturm, A. and Shehory, O.: A Framework for Evaluating Agent-Oriented Methodologies. In: International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'03). LNCS. Vol. 3030. pp.: 94-109. Springer-Verlag, 2003.
- [SUN, 2005] SUN: Java Remote Method Invocation (Java RMI). Available at <http://java.sun.com/products/jdk/rmi/>, 2005. Last access in December 2005.
- [Susi et al., 2005] Susi, A., Perini, A., Mylopoulos, J., Giorgini, P.: The Tropos Metamodel and its Use. Informatica, Slovenia, 29(4), pp. 401-408, 2005.
- [Tahara et al., 1999] Tahara, Y., Ohsuga, A., Honiden, S.: Agent system development method based on agent patterns. In: 21st International Conference on Software Engineering, Los Angeles, California, United States. ISBN: 1-58113-074-0, pp.: 356 – 367, 1999.
- [Tarr et al., 1999] Tarr, P.; Ossher, H.; Harrison, W. and Sutton Jr, S.: N Degrees of Separation: Multidimensional Separation of Concerns. In: 21st International Conference on Software Engineering (ICSE'99), pp.:107–119, May 1999.
- [Wagner, 2003] Wagner, G.: The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior. In: Information

- Systems, 28, 5, pp. 475 – 504, 2003.
- [Wautelet et al., 2005] Wautelet, Y., Kolp, M. and Achbany, Y.: S-Tropos - An Iterative SPEM-Centric Software Project Management Process. Technical Report, Available at http://www.isys.ucl.ac.be/staff/youssef/Articles/WP_SPEM.pdf, 2005. Last access in December 2006.
- [Weiss, 2000] Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. pp.: 27-78, ISBN: 0262731312, 2000.
- [Whittle and Araújo, 2004] Whittle, J. and Araújo, J.: Scenario Modeling with Aspects. In: Rashid, A., Moreira, A., Tekinerdogan, B. (Eds.), IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, August 2004.
- [Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. R.: Intelligent agents: theory and practice. The Knowledge Engineering Review, Vol. 10. Issue 2. pp.: 115-152, 1995.
- [Wooldridge, 2002] Wooldridge, M.: An Introduction to Multiagent Systems. England. pp.: 15-103, ISBN: 0-471-49691-X, 2002.
- [Yu, 1995] Yu, E.: Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.
- [Yu, 2002] Yu, E.: Agent-Oriented Modelling: Software Versus World. In: Wooldridge, M., Weiss, G. and Ciancarini, P. (Eds.), Agent-Oriented Software Engineering (AOSE'01), LNAI, Vol. 2222, Springer-Verlag, pp.: 206-225, 2002.

- [Zambonelli et al., 2003] Zambonelli, F., Jennings, N. R. and Wooldridge, M.: Developing Multiagent Systems: the Gaia Methodology. In: ACM Transactions on Software Engineering and Methodology, Vol. 12, Issue 3, pp.: 317-370, 2003.

Appendices

1 Case Study I – e-News System

1.1 Early Requirements

The Strategic Dependency (SD) model depicted in Figure 2.1 focuses on the intentional relationships among organizational actors. It consists of a set of nodes and links connecting them, where nodes represent actors and each link indicates a dependency between two actors. The depending actor is called *depender* (e.g., *user*), and the actor who is depended upon is called the *dependee* (e.g., *Webmaster*). Hence, an SD model consists of a network which captures the motivation and the rationale of activities. The i* framework distinguishes four types of dependencies. In a goal dependency (*Read news, Newspaper edited according to the guidelines, Guideline meeting coordinated, Published Newspaper, Guideline of specific subject provided*), an agent depends on another to fulfill a goal, without worrying how this goal will be achieved. In a resource dependency (*News, Photography, Newspaper Guideline, Suggested news to publish, Newspaper, Guideline of specific subject, Specific category guideline, Specific category news, Specific category photography*), an agent depends on another agent to provide a physical resource or information. In a task dependency (*Make photographic covering of news*), an agent depends on another to carry out a task.

A softgoal dependency (*Availability[Newspaper], Updated[Newspaper], Quality [Photography], Original [News]*) is similar to a goal dependency, except that the satisfaction of a softgoal is not precisely defined. In i* we can also classify actors into agents, roles and positions. An agent is an actor with concrete physical manifestations (a person or a system). A role (User, Editor, Editor in Chief, Photographer, Reporter, Webmaster, Chief Editor) is an abstract characterization of the behaviour of a social actor within some specialized context, domain or endeavor. A position is a set of roles typically played by one agent.

Finally, i^* provides the Strategic Rationale (SR) model (Figure 2.2) for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors by using a means-ends analysis. From the means-ends analysis of the Editor in Chief actor, we have identified the necessity of a computational system in order to automate the edition and publication of the newspaper on website and, therefore, decrease editing and publishing costs. Thus, the e-News system comes into the picture as one actor who contributes to the fulfillment of stakeholder goals (Figure 1).

1.2 Late Requirements

The e-News system (Figure 1) allows the User to read specific news which can be searched by providing a keyword or read news provided by the newspaper. Thus, the User depends on e-News System to read news while the e-News System depends on User get a keyword in order to enable the search of the specific requested news. Both the User and Editor in Chief expects that e-News system maintain newspaper updated and available on website. The Editor in Chief depends on e-News system for editing and publishing newspaper according to guideline and also expects that e-News system be adaptable, interoperable, maintain published information secure and publish original news. The newspaper editing and publishing must be according to a guideline, which is produced by a meeting among the Editors and the Editor in Chief. Since Editor in Chief is a special kind of Editor, it can also suggest news to be included in the guideline. The Editor in Chief is the coordinator of the guideline meeting and therefore, the e-News System depends on him to get the newspaper guideline. To compose the newspaper it is necessary for the e-News System to interact with different and distributed news' agencies on the Internet, in order to get news according to the guideline. Thus, the e-News System depends on News Agency to get news and photos which have their data integrity guaranteed. After newspaper edition is concluded, the e-News system depends on Editor in Chief to have the newspaper publishing authorized.

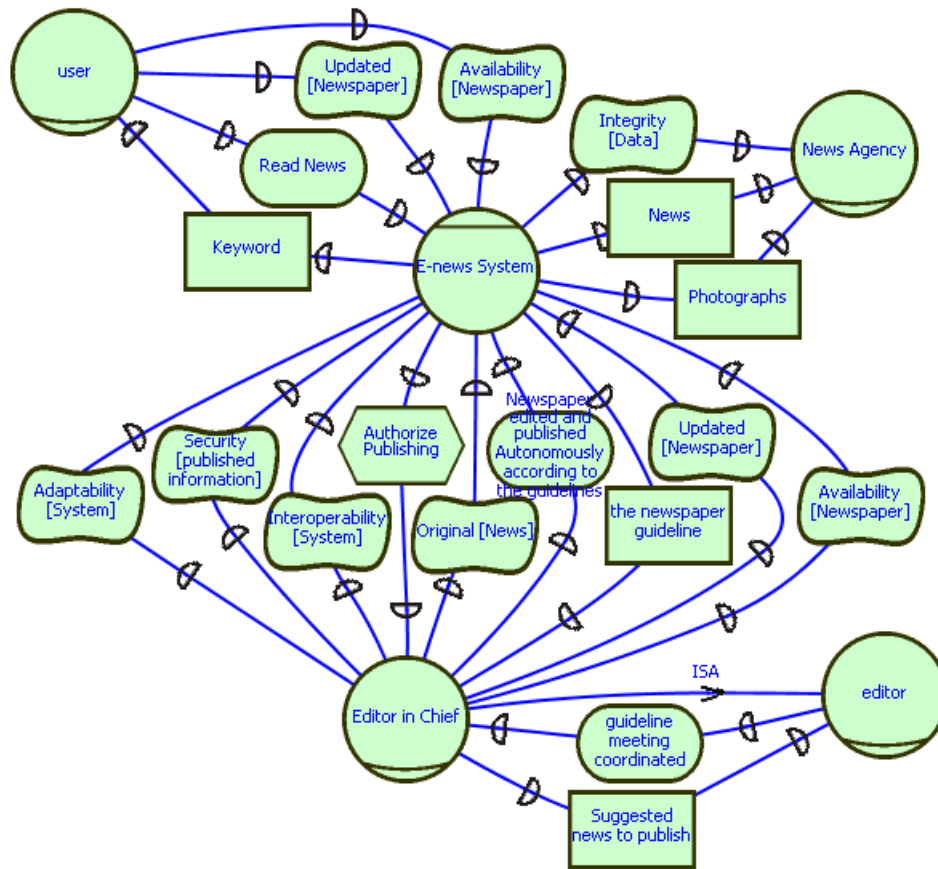


Figure 1 – Strategic Dependency Model for the e-News System

The SR model (Figure 2) focuses on one of the goal dependency identified for *User*, namely *Read News*. To achieve that goal, the analysis postulates a *Published News* goal that can be fulfilled by means of two goals alternatives: *Published News Searched by Keyword* or *Published Newspaper According to Guideline*. The former goal is fulfilled by means of a *Publish News Searched by Keyword* task, while the latter goal is fulfilled by means of a *Publish Newspaper According to Guideline* task. Tasks are partially ordered sequences of steps intended to accomplish some (soft)goal. Tasks can be decomposed into goals and/or subtasks, whose collective fulfillment completes the task. In the figure, *Publish News Searched by Keyword* task is decomposed into *Search News by Keyword* and *Release Searched News on Website* tasks which together accomplish the top-level task. The *Publish Newspaper According to Guideline* task is decomposed into *Available [System]*, *Interoperable [System]*, *Adaptable [System]* and *Secure [System]* softgoals, *Decompose the Guideline*, *Edit News Article for each Category*, *Edit the Newspaper* and

Update Newspaper on Website tasks, and *Evaluated Newspaper Suitability* goal which together accomplish the top-level task.

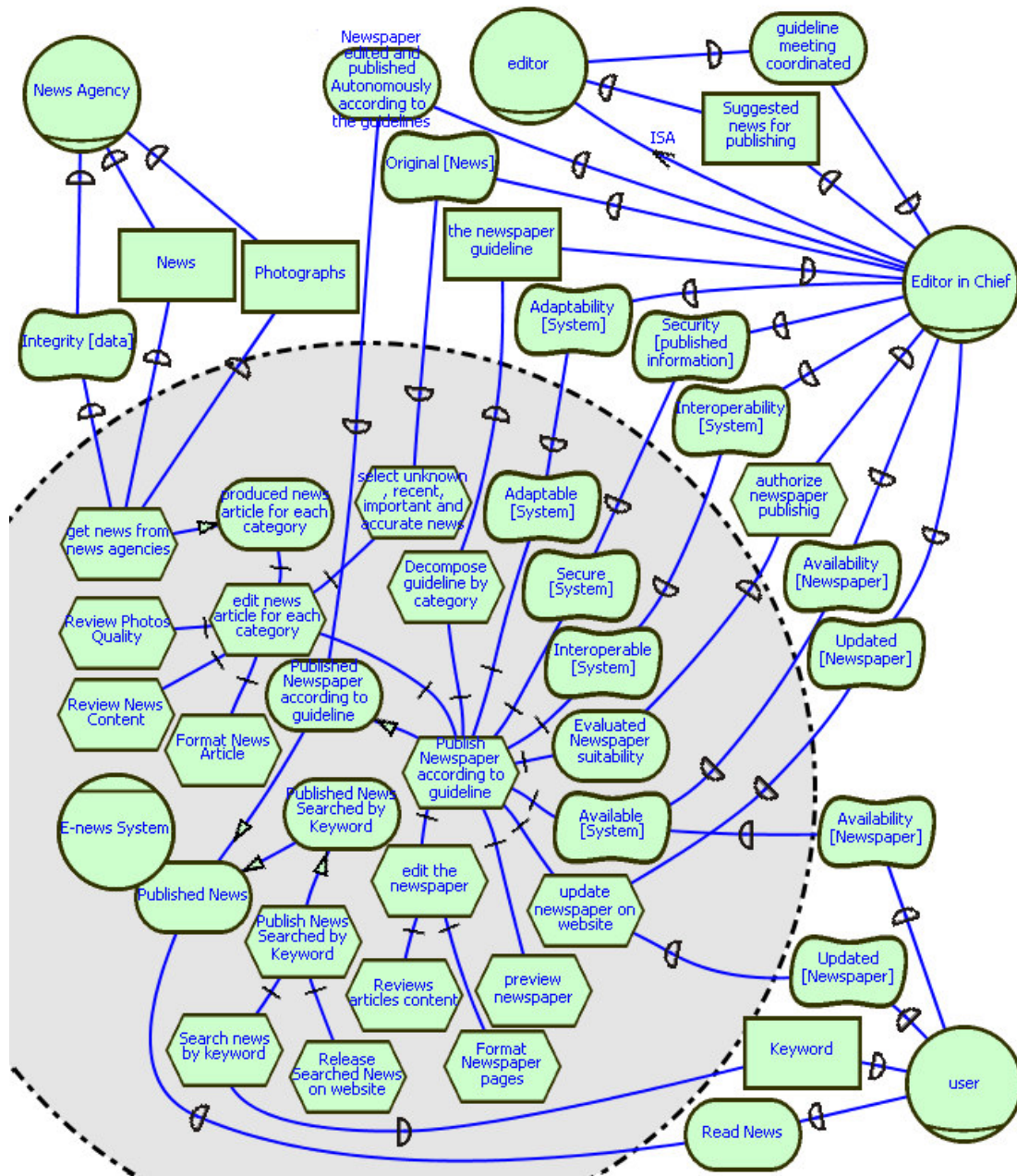


Figure 2 – Strategic Rationale Model for the e-News System

Sub-goals and subtasks can be specified more precisely through refinement. For instance, the *Edit the Newspaper* task is fulfilled through *Review Articles Content*, *Format Newspaper Pages* and *Preview Newspaper* tasks while the *Edit News Article for each Category* task would be collectively accomplished by *Produced News Article for each*

Category goal and Select Unknown, Recent, Important and Accurate News, Review Photo Quality, Review News Content and Format News Article tasks. The *Produced News Article for each Category* goal is fulfilled by means of a *Get News from News Agencies* task. These decompositions eventually allow us to identify actors who can accomplish a goal, carry out a task, or deliver on some needed resource for *e-News System*. Such dependencies in the Figure 2 are the resource and softgoal dependencies on *News Agency* for providing news and photos with data integrity, the resource and task dependencies on *Editor in Chief* for providing the newspaper guideline and performing authorization of newspaper publishing, and the resource dependency *Keyword* on *User* to perform the searching of a specific news.

1.3 Architectural Design

The first task during architectural design is to select among alternative architectural styles using for example, the desired qualities earlier identified, namely *Available [System]*, *Adaptable [System]* and *Secure [System]* softgoals. To conduct such quality analysis, Tropos uses the Correlation Catalogue [Kolp et al., 2001a] [Kolp et al., 2001b] which summarizes the evaluation of organizational styles in the fulfillment of quality attributes. Eventually, the analysis from Table 7.2 allows us to choose the *Joint Venture* architectural style for our case study.

Assigning the operationalization of system requirement to each architectural agent role according to the Join Venture Style can be achieved following the approach presented in [Bastos, 2004]. In fact, the separation of concerns at architectural level complies to design principles, such as low coupling and high cohesion. In this work version, we perform the assignment of system's responsibility in an ad hoc manner. Figure 3 shows the grouping of responsibilities to be assigned to the each agent role composing the multi-agent system, as well as the identification of these agent roles.

Figure 8.1 shows a multi-agent system architecture in *i** for an *e-News* system that applies the joint-venture style [Kolp et al., 2002] at the architectural design level. In a *joint venture* organisational style, a specific joint management actor (e.g. Chief Editor) coordinates tasks and manages the sharing of resources between partner actors (e.g.

Editor, Webmaster, Reporter and Photographer). Each partner can manage and control itself on a local dimension and may interact directly with other partners to exchange resources, such as data and knowledge. However, The Chief Editor is the only responsible for the strategic operation and coordination of such system, and its actors on a global dimension.

Table 1 – Evaluation of Correlation Catalogue

Quality Attributes									
Styles	Predictability	<i>Security</i>	<i>Adaptability</i>	Cooperativity	Competitivity	<i>Availability</i>	Integrity	Modularity	Aggregability
Flat Structure	--	--	-			+	+	++	-
Structure-in-5	+	+		+	-	+	++	++	++
Pyramid	++	++	+	++	-	+	--	-	
Joint-Venture	+	+	++	+	-	++		+	++
Bidding	--	--	++	-	++	-	--	++	
Takeover	++	++	-	++	--	+		+	+
Arm's-Length	-	--	+	-	++	--	++	+	
Hierarchical Contracting			+	+	+	+		+	+
Vertical Integration	+	+	-	+	-	+	--	--	--
Co-optation	-	-	++	++	+	--	-	--	

According to Figure 8.1, when an user wishes to read news, she accesses the newspaper website maintained by a Webmaster agent which is responsible for updating the published information. The information to be published is provided and authorised by the Chief Editor role. The Chief Editor role depends on the Editor role to have the news prepared according to the newspaper guideline.

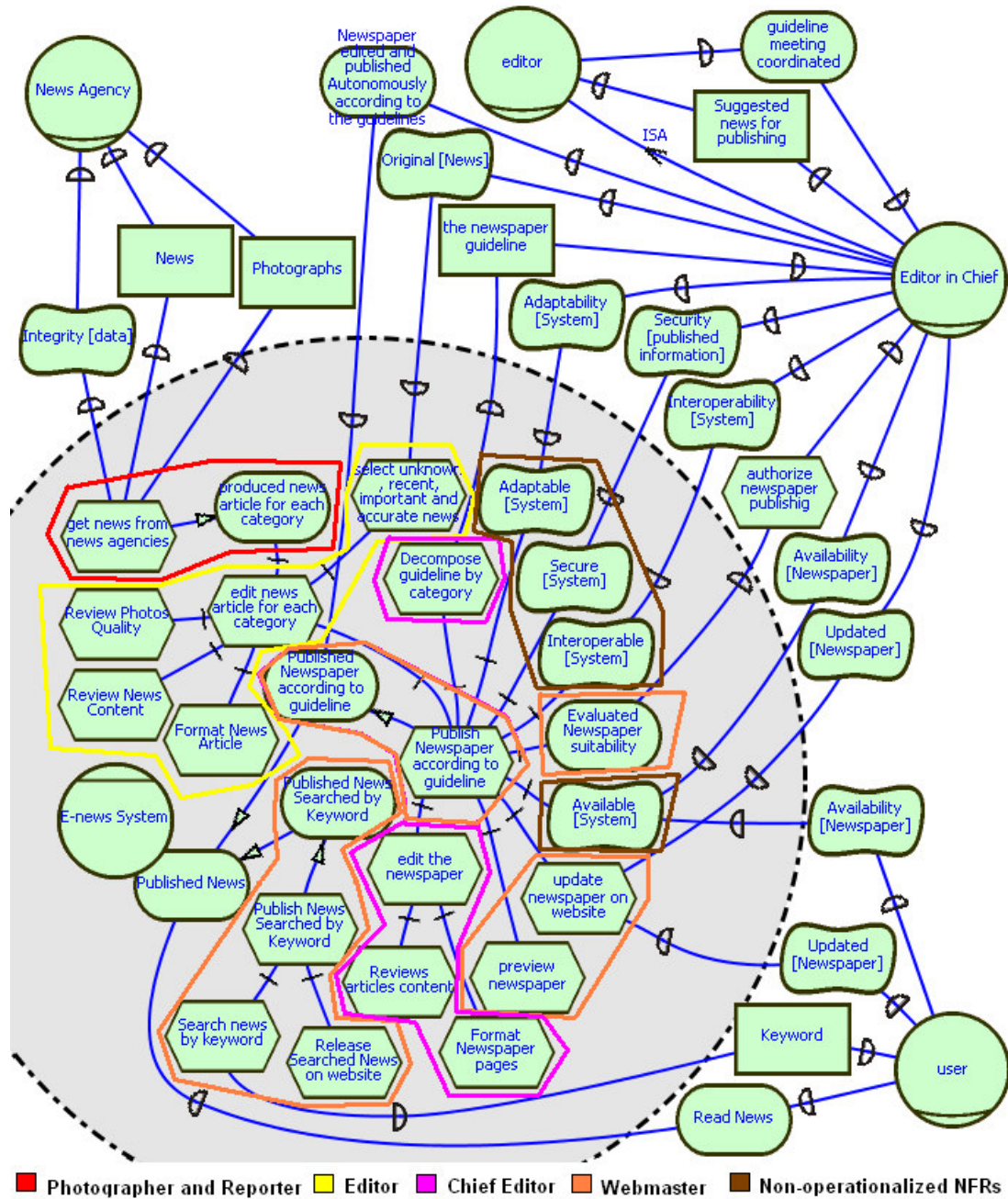


Figure 3 – Assignment of the e-News System Responsibilities

Thus, it is necessary for the Chief Editor to share the newspaper guideline among the Editors, which are responsible for getting news of a specific type. For example, an Editor may be responsible for political news while another one may be responsible for sports news. Each Editor selects one or many reporters and photographers which can find the news of specific subjects (e.g., basketball) to fulfill its sub-guideline (e.g., about sport

news). The Chief Editor then edits the Editor' news and forwards them (if authorized) to the Webmaster to publish them.

2 Case Study II - Conference Management System

Now let us consider the domain of conference management introduced in [Zambonelli et al., 2003] and modeled using the Tropos framework in [Silva, C. et al. 2006a]. A conference involves several individuals. During the submission phase, *Authors* submit papers, and are informed that their papers have been received and have been assigned a submission number. In the review phase, the *Chair* has to handle the review of the papers by contacting potential *Reviewers* and asking them to review a number of papers according to their expertise. Eventually, reviews come in and are used to decide about the acceptance or rejection of the submissions. In the final phase, *Authors* need to be notified of these decisions and, in case of acceptance, will be asked to produce and submit a revised version of their papers. The *Publisher* has to collect these final versions and print the proceedings. Figure 4 presents the Conference Management System, a solution developed as an example of MAS for the conference management domain. The structure-in-5 architectural style [Kolp et al. 2002] has been chosen and applied to the MAS architectural design, but due to lack of space we do not show how we made the choice. Our focus is on the design of the MAS architecture according to agent-oriented modeling diagrams (Chapter 5).

The initial version of the Conference Management System supports the submission, review and notification phases of the conference process. The Conference Management System architecture is decomposed into four actors: Submission Manager, Review Manager, Notification Manager and Reviewer. Each of these actors is linked to the system through an *is-part-of* relationship (see Figure 4). The Submission Manager is responsible for handling the submission phase of the process. The Review Manager is responsible for distributing the set of submitted papers to at least n reviewers according to their research area. The Notification Manager is in charge of handling the notification phase process. The Reviewer actor is responsible for evaluating a paper proposal according to the reviewer preferences and skills.

Figure 4. Conference Management System Architecture

2.1 From i* to UML

The process presented in Section 7.1. is going to be used to produce MAS UML-based models at the architectural level in the context of Tropos. We begin by performing the means-end analysis for each actor which belongs to the MAS architecture described using the i^* notation. Then, we rely on the mapping heuristics (Section 7.1.1) to specify each diagram presented in Chapter 5. For example, in our case study we perform the specialized means-end analysis of the Reviewer, Review Manager, Notification Manager and Submission Manager actors in order to capture their rationale when pursuing their goals and dependencies. The Review Manager expects to have *Papers Reviewed*. One

alternative to satisfy this goal is to perform the *Manage Review Phase* task. This task is decomposed into four sub-tasks (see the refined model in Figure 5): *Collect Papers Review*, *Select "n" Reviewers of Paper Research Area*, *Propose Paper Review* and *Assign Paper Reviewer*.

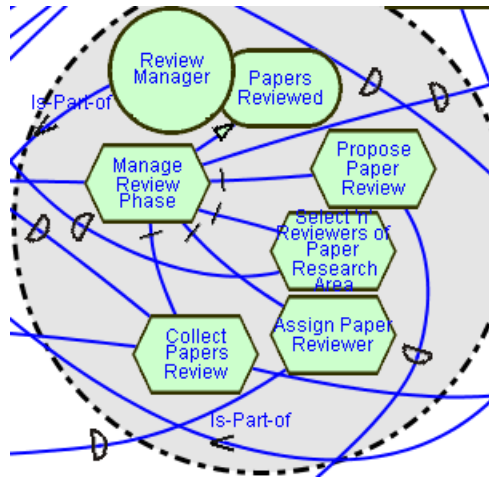


Figure 5. Means-end analysis for Review Manager

Analogously, the Reviewer actor expects to get hold of *Proposal for Review Evaluated* and, to achieve this goal, it has the alternative of performing the *Evaluate Proposal for Review* task. To carry out this task it is necessary to perform several subtasks: *Evaluate Relevance of Conference*, *Evaluate Time Availability*, *Evaluate Interest in Paper Subject* and *Set Personal Profile*. The Submission Manager actor is in charge of having the *Papers Submission Managed* and to accomplish this goal it has one alternative which is performing the *Manager Submission Phase* task. This task is decomposed into the *Assign Submission Number* and *Collect Paper Submission* sub-tasks. The Notification Manager actor expects to obtain the *Papers Notification Managed* and to reach this goal it has one alternative which is performing the *Manager Notification Phase* task. This task is decomposed into the *Notify Authors* and *Collect Revised Version of Accepted Papers* sub-tasks. The means-end analysis models of the Reviewer, Submission Manager and Notification Manager actors have been omitted here due to the lack of space.

2.2 Architectural diagram

Having concluded the means-end analysis of Reviewer, Submission Manager, Notification Manager and Review Manager actors, we can now move on to identifying

the properties that characterize that agent according to the MAS modeling diagrams (Chapter 5). The heuristics presented at Section 7.1.1 can be of some assistance to describe the Reviewer, Submission Manager, Notification Manager and Review Manager actors according to the architectural diagram (Figure 5.4). For example, the *Papers Reviewed* goal present in the means-end analysis of the Review Manager actor becomes a «Goal» associated to the Review Manager «Agent» class (colored area of Figure 6). The *Manage Review Phase* task becomes a «Plan» associated to both the Review Manager «Agent» class and Papers Reviewed «Goal» class. Each of the *Collect Papers Review*, *Select "n" Reviewers of Paper Research Area*, *Propose Paper Review* and *Assign Paper Reviewer* tasks becomes an «AgentAction» in the Manage Review Phase «Plan» class.

Each dependum of the Review Manager actor's dependencies becomes a «Dependum» interface. For example, the dependum of the *Submitted Papers* resource dependency, between Review Manager and Submission Manager agents, becomes a *Submitted Papers* «Dependum» interface, typed as a resource (colored area of Figure 6). The depender and dependee roles of the *Submitted Papers* dependency become the «Depender» dependency from Review Manager «Agent» class and the «Dependee» realization from Submission Manager «Agent» class, respectively. The *Submitted Papers* resource dependency itself becomes a «Connector» between Review Manager and Submission Manager «Agent» classes. Ports are added to these «Agent» classes to enable the link through the «Connector». The same guidelines are applied to the other actors present in the i* model. As a result several classes are incorporated in the MAS architectural diagram depicted in Figure 6.

The architectural design of the Conference Management system (Figure 6) is performed by using the MAS architectural pattern depicted in Figure 5.4 to assign the system's responsibilities to the architectural components. The Conference Management system is composed of four agents: Submission Manager, Review Manager, Reviewer and Notification Manager. For example, in Figure 6 the colored area corresponds to the interaction between the Review Manager and Submission Manager agents to achieve the service Submitted Papers. The Review Manager agent intends to achieve the Papers Reviewed goal by means of the Manage Review Phase plan. However, to get the papers

reviewed the Review Manager agent has to request the Submission Manager agent to perform the Submitted Papers service. This service provides to the Review Manager agent the submitted papers collected by the Submission Manager agent. The Submission Manager performs the requested service because it does not conflict with the achievement of the Submission Phase Managed goal. Hence, both the requested service and the goal achievement are accomplished by means of the Manage Submission Phase plan. The description of the Proposal Review Accepted and Papers Review services is achieved in a similar way.

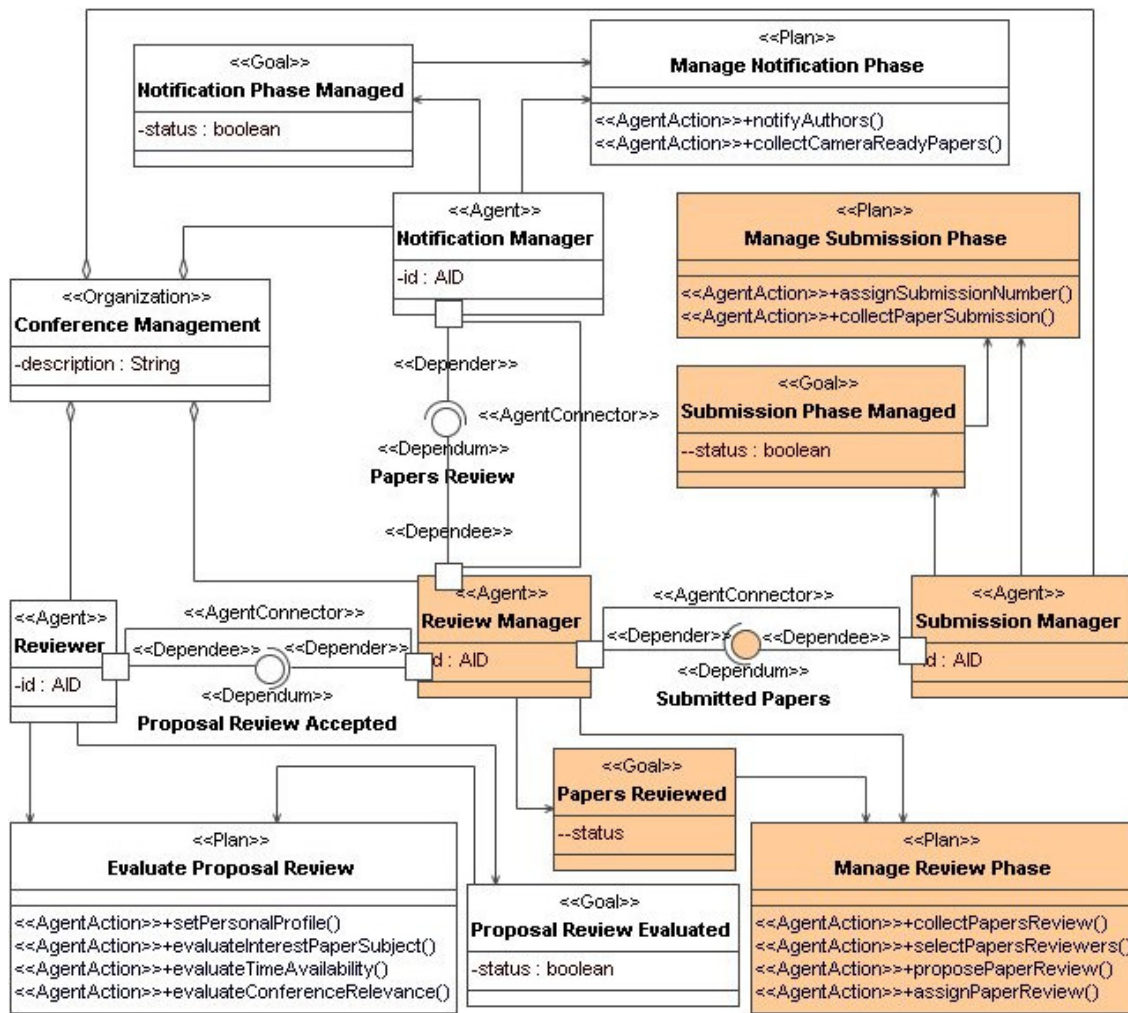


Figure 6. Conference Management Architectural Diagram

2.3 Communication diagram

The communication diagram is defined in terms of instances of agents and the messages exchanged between them to achieve their responsibilities. For example, Figure 7 shows an interaction involving the Notification Manager, the Review Manager, the Reviewer and the Submission Manager agents.

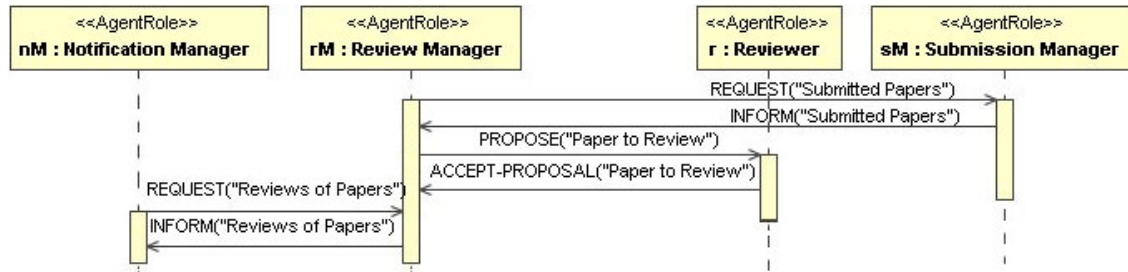


Figure 7. Conference Management Communication Diagram

The interaction specified using the communication diagram is asynchronous. Hence, the Review Manager agent sends a message requesting the submitted papers which are going to be provided through the execution of Submitted Papers service. Then, the Submission Manager answers by informing if the requested service has been performed successfully. The Review Manager agent sends a message to the Reviewer agent proposing a paper to review. If the Reviewer agent answers by accepting the proposal to review the paper, then the Proposal Review Accepted service is assumed to be achieved. The Notification Manager agent, in turn, sends a message to the Review Manager agent requesting the reviews of the papers which are going to be provided through the execution of Papers Review service. Then, the Review Manager answers by informing if the requested service has been performed successfully.

2.4 Environmental diagram

The environmental diagram is defined in terms of agents composing an organization which is situated in an environment composed by resources which are accessed by the agents according to its rights. The heuristics presented in Section 7.1.1 continue to be used here to identify the properties which characterize that agent according to this diagram. Hence, all *Submitted Papers*, *Papers Review*, *List of Reviewers* resource elements related to the Review Manager actor (Figure 5) become a «Resource» associated to the Review Manager «Agent» class in the environmental diagram presented

in the Figure 8. In this diagram we have the *Review Manager* and *Submission Manager* agents composing the *Conference Management* organisation which is situated in the *Conference Management* environment. The Notification Manager and Reviewer agents have been omitted here due to the lack of space.

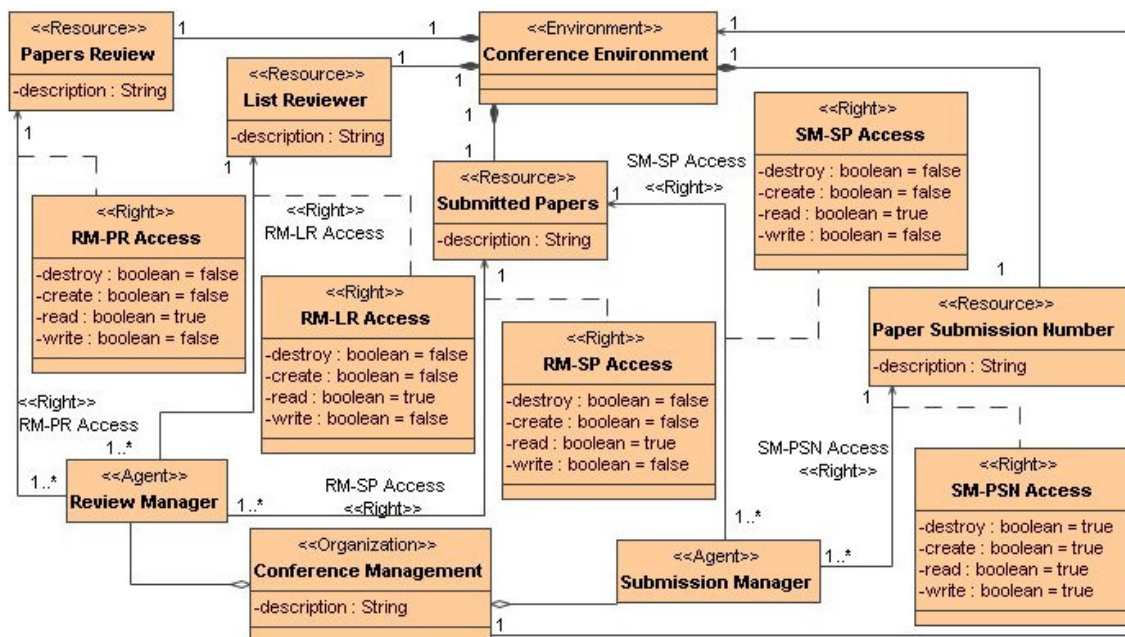


Figure 8. Conference Management Environmental Diagram

The *Submission Manager* agent needs to access the *Submitted Papers* resource available at the *Conference Management* environment to perform the Manage Submission Phase plan. The *Submission Manager* agent can only read the *Submitted Papers* resource, according to its *SM-SP Access* right (read *Submission-Manager- Submitted Papers Access* right). The *Review Manager* agent needs to access a *Papers Review* resource available in the *Conference Management* environment to perform the Manage Review Phase plan. The *Review Manager* agent can only read the *Papers Review* resource, according to its *RM-PR Access* right (read *Review Manager-Papers Review Access* right). The *Review Manager* agent also needs to access a *Submitted Papers* resource to perform the Manage Review Phase plan. His permission is for only reading the *Submitted Papers* resource, according to its *RM-SP Access* right (read *Review Manager-Submitted Papers Access* right). The description of the agent's rights to access the other resources is achieved in a similar way.

2.5 Intentional diagram

The intentional diagram is defined in terms of agents, their beliefs, goals, plans, norms and ontology. The heuristics presented in Section 7.1.1 are used here to identify the properties which characterize that agent according to this diagram. Hence, the condition to perform the *Collect Papers Review* task (Figure 5) is that the Review Deadline should have passed. The circumstance to perform the *Select "n" Reviewers of Paper Research Area*, *Propose Paper Review* and *Assign Paper to Reviewer* tasks is that the Submission Deadline should have passed. Each of these conditions becomes a «Belief» associated to the Review Manager «Agent» class. However, the intentional diagram is not shown because we have not defined yet the heuristics to derive both the MAS ontology and norms. These issues will be addressed in future work.

3 Catalogue of Social Patterns

In this section we provide a catalogue of mediation social patterns [Kolp et al., 2005] described according to the APS technique. The Intent, Motivation Example, Applicability, Participants and Solution fields of the following pattern templates are based on [Hayden and Yang, 1999].

3.1 Embassy

The Embassy social pattern template is depicted in Table 11.2.

Table 2 – The Template for Embassy Pattern

Element	Description
<i>Name</i>	Embassy Pattern
<i>Intent</i>	Multi-agent systems may use an embassy to represent them, through which a foreign agent may communicate and gain access. The embassy provides access control, and so should feature in the design of a security mechanism for agents. The embassy may also allow translation and the establishment of communication protocols where the foreign agent is not

	homogeneous.
<i>Applicability</i>	If an application makes occasional reference to another application in the same environment, but it is not desirable to permanently integrate or interface these applications, an embassy is a suitable alternative. If the agents in both domains speak the same ACL, embassy facilitated interaction is possible. Otherwise more effort must be spent translating between ACLs.
<i>Motivation Example</i>	An agent environment such as the Web, may comprise many different agent domains, such as an e-commerce agent, information retrieval agents and chatterbots, in the same Web space. These domains may be entirely independent or may wish to interact at some point e.g. an information agent wishes to query the commercial agent regarding a specific stock price. The embassy agent aims to bridge these separate agent domains, allowing interaction between heterogeneous agent applications. The embassy agent may need to translate different communication languages (ACLs) and content formats. Standardization and ubiquity of an ACL will make the embassy's task easier.
<i>Problem</i>	How can agents from heterogeneous agent applications interact?
<i>Solution</i>	A foreign agent or embassy requests access to an agent domain from its embassy agent. Depending on the level of the digital certificate provided, access may be granted or denied. The foreign agent may then submit messages to the embassy for translation. The content of the message is translated in accordance with a standard ontology. A library of such ontologies would need to be available. Translated messages must then pass through another level of security before being forwarded to the domain agents. The results of the translated query/message are passed back out to the foreign agent, translated in reverse. The interaction may be discontinued by the embassy, any of its domain agents or the foreign

	agent at will.
<i>Participants</i>	This coordination pattern involves at least one embassy, foreign and local/native agents.

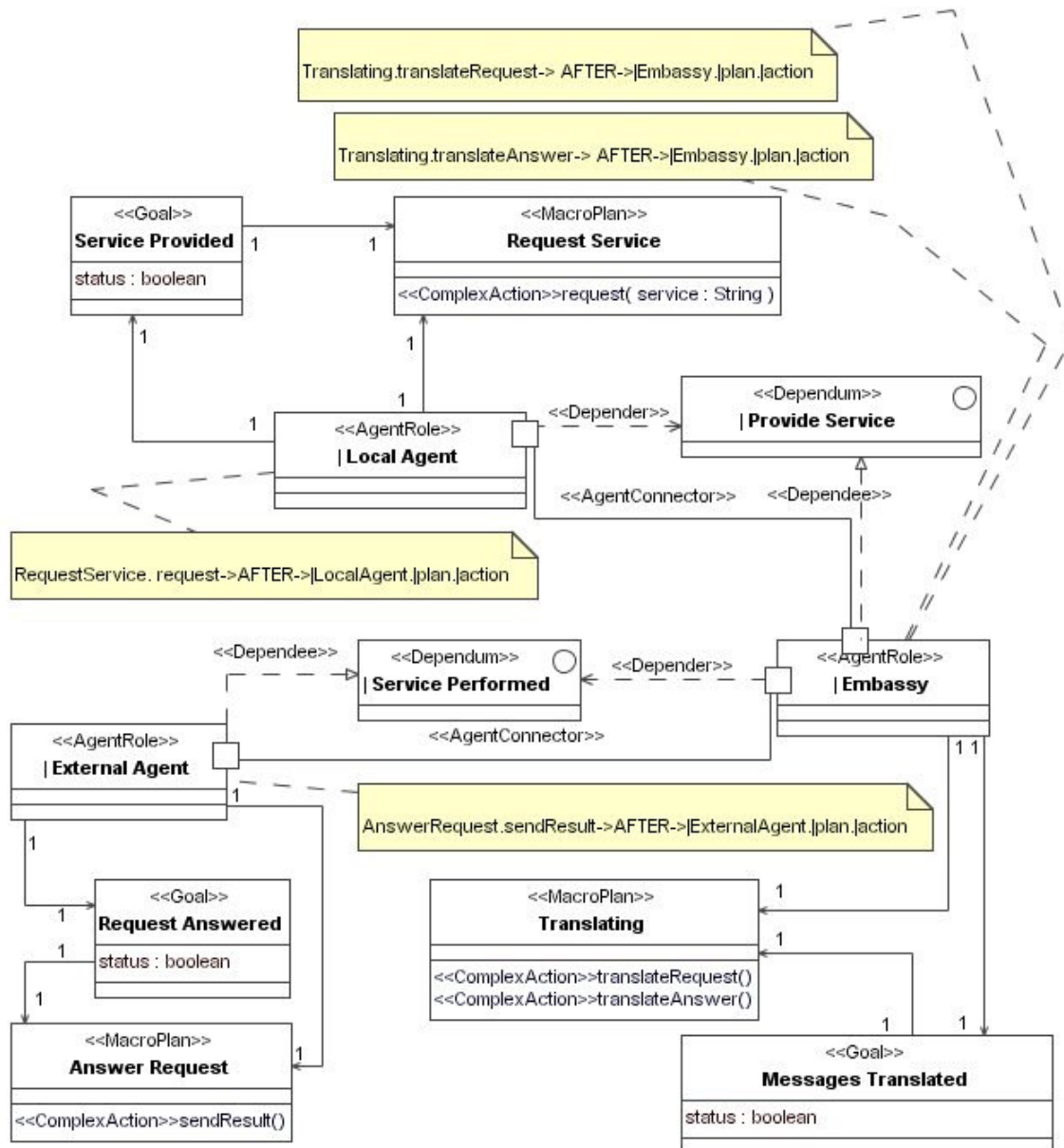


Figure 4 – Embassy Structural Agent Pattern Specification

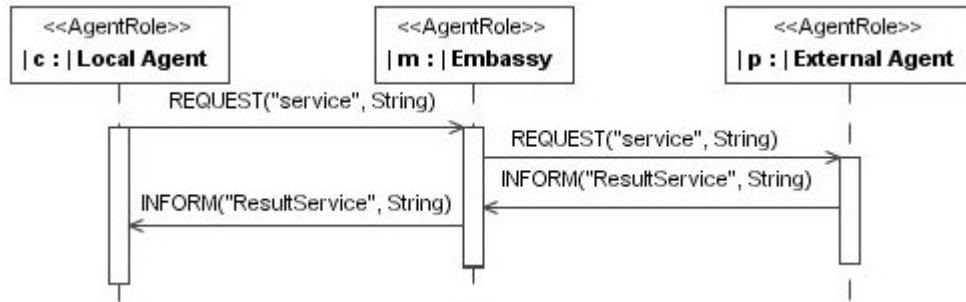


Figure 5 – Embassy Communication Agent Pattern Specification

3.2 Mediator

The Mediator social pattern template is depicted in Table 3.

Table 3 – The Template for Mediator Pattern

Element	Description
<i>Name</i>	Mediator Pattern
<i>Intent</i>	This pattern enables the encapsulation of agent interactions i.e. the specification of how a set of agents interact. The nature of the interactions can then be maintained at only one point, the mediator.
<i>Applicability</i>	A mediator pattern is suitable if interactions between the agents are complex, but well-defined; customization of distributed behaviors is desired; or agent reuse is hampered because the agent communicates with too many other agents.
<i>Motivation</i> <i>Example</i>	If behaviors are distributed among agents, many interactions are possible. Agents may need to hold both self models and acquaintance models of all other agents to support all possible interactions. Efficiency is reduced as interconnections proliferate, as agents become reliant on too many other agents. The mediator pattern encapsulates collective behavior, allowing control and coordination of the interactions of a group of agents. It introduces a layer of abstraction, preventing agents from referring to each other explicitly, and allowing their interactions to

	be varied independently. This promotes loose coupling and reduces the number of interconnections directly between the agents, since they only need to know about the mediator. Maintenance of behaviors is also easier when localized in a mediator, as opposed to behaviors which are distributed.
<i>Problem</i>	How encapsulate, control and encapsulate interactions of a group of agents?
<i>Solution</i>	Colleagues send and receive requests from the Mediator, which routes requests between appropriate colleagues in accordance with a specified conversation model.
<i>Participants</i>	This coordination pattern involves a mediator, and any number of colleague agents. The mediator coordinates the cooperative behavior of the colleagues and has acquaintance models of all colleague agents. The colleagues each have an acquaintance model of the mediator. A colleague agent addresses the mediator in place of another colleague.

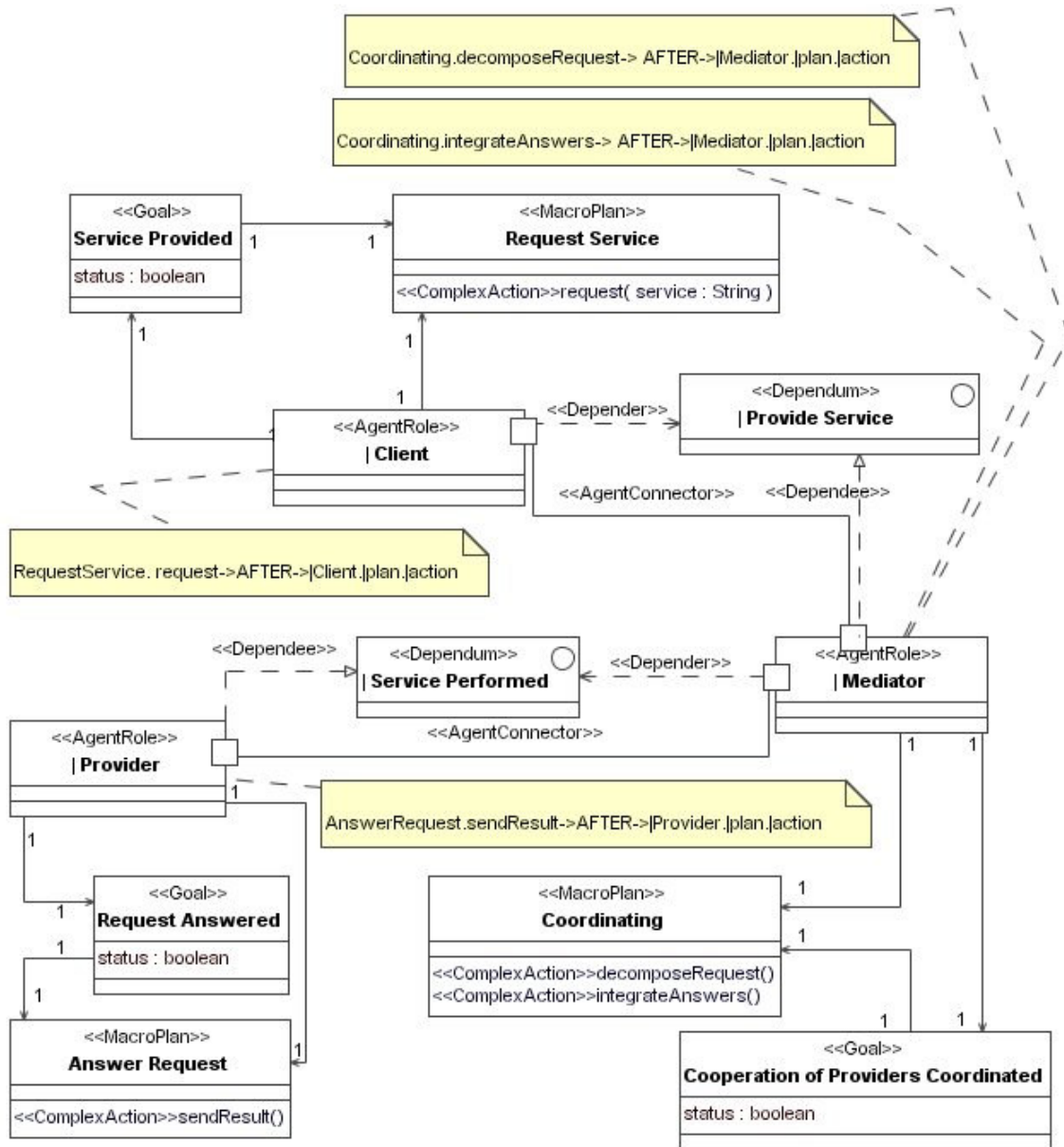


Figure 6 – Mediator Structural Agent Pattern Specification

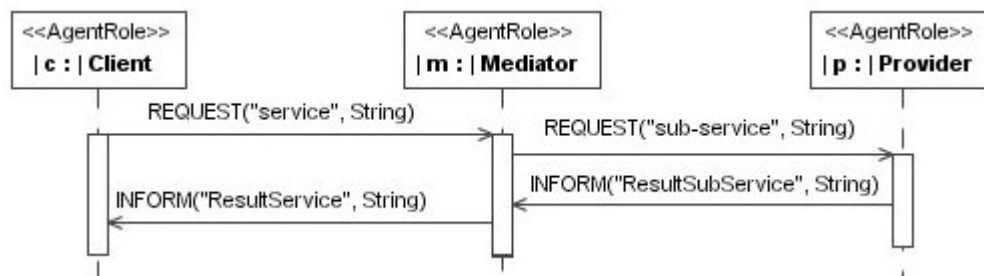


Figure 7 – Mediator Communication Agent Pattern Specification

3.3 Monitor

The Monitor social pattern template is depicted in Table 11.4.

Table 4 – The Template for Monitor Pattern

Element	Description
<i>Name</i>	Monitor Pattern
<i>Intent</i>	This pattern monitors subjects of interest for state changes and notifies agents of their occurrence..
<i>Applicability</i>	The monitor pattern is suitable if an indefinite number of agents are dependent on a subject of interest or it is desirable that the subject of interest be decoupled from the dependent agents who are to be notified of its changes.
<i>Motivation Example</i>	Maintaining consistency in a distributed system is a challenge. Tight coupling preserves consistency, but compromises decoupling and reusability. In this pattern, agents which must remain current with respect to a subject advise the monitor of interest in the subject by subscribing for notification of particular changes to that subject. The monitor then requests notification of state changes from the subject. When the subject changes it notifies the monitor of its new state information. The monitor then notifies subscriber agents for whom the state change is relevant. This arrangement maintains consistency whilst decoupling monitor, subscribers and subjects.
<i>Problem</i>	How can an agent dependent on another agent can be decoupled from the depended agents and, even so, be notified of its changes?
<i>Solution</i>	Subscription, requests for notification, the actual notification and revoking subscription and requests for notification are all required by

	this pattern.
<i>Participants</i>	This coordination pattern involves at least one monitor, a number of subscriber agents and at least one subject or event of interest. The role of the monitor is to accept subscriptions, request notifications from subjects of interest, receive such notifications of events and to alert subscribers to relevant events. The role of the subject is to provide notifications of state changes as requested. The role of the subscriber is to subscribe for notifications of state changes to distributed subjects (data or objects), receive notifications with current state information, and update its local state information. A subject may have many subscribers, each of which may express interest in different state changes. Subscribers are notified when their event or state change of interest occurs. The notification includes all state information required to update the subscriber and thereby maintain consistency with the subject's state information.

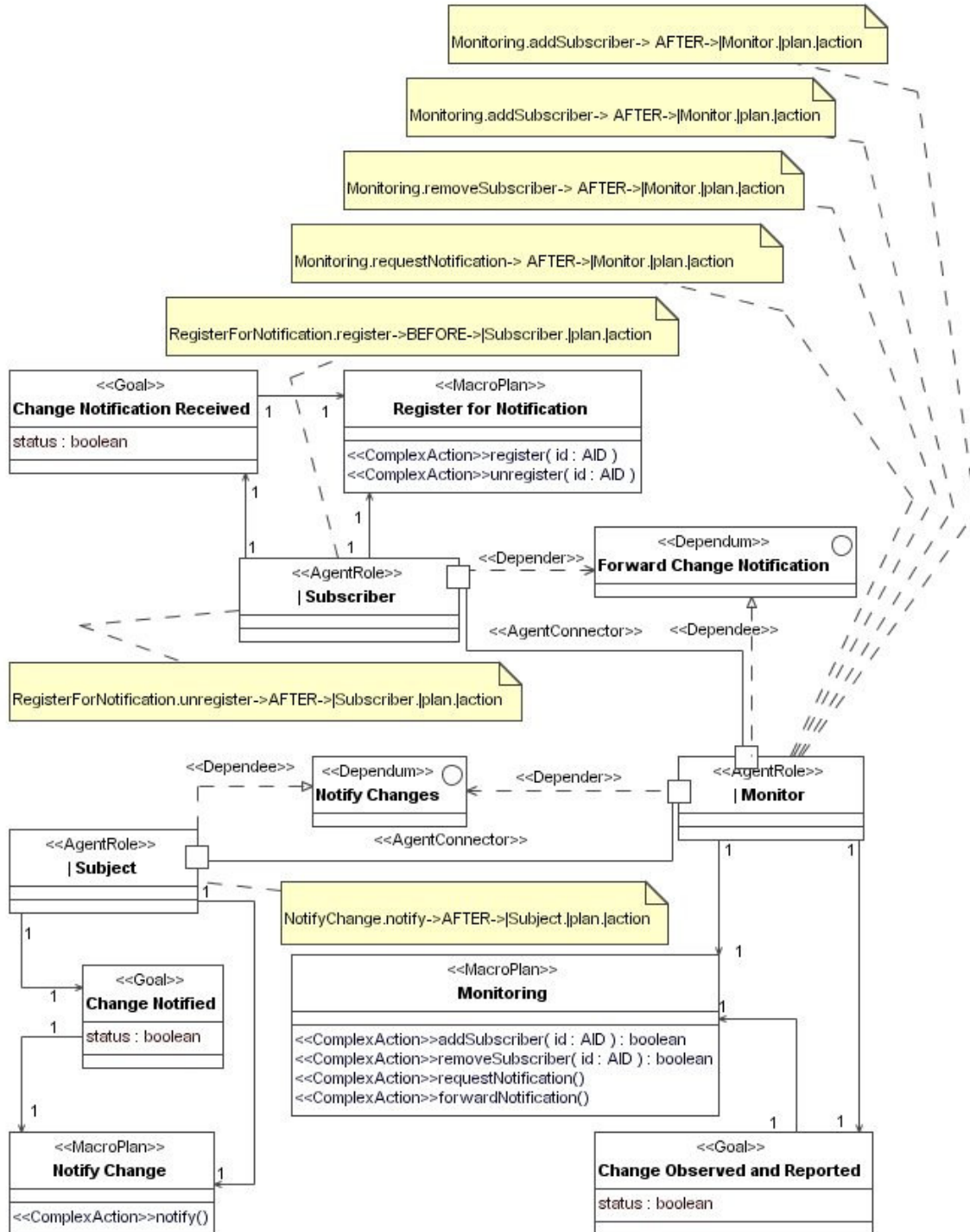


Figure 8 – Monitor Structural Agent Pattern Specification

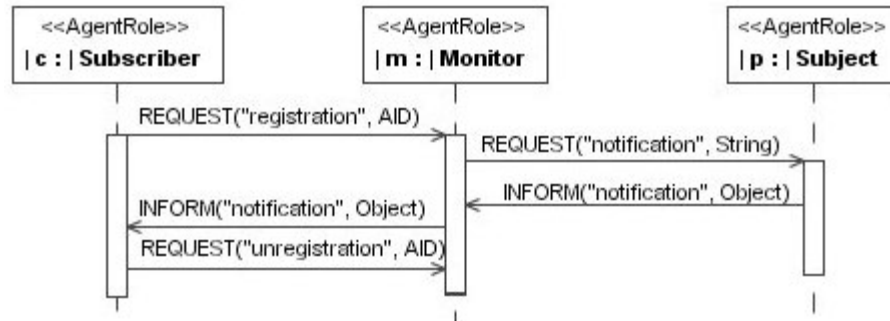


Figure 9 – Monitor Communication Agent Pattern Specification

3.4 Wrapper

The Wrapper social pattern template is depicted in Table 5.

Table 5 – The Template for Wrapper Pattern

Element	Description
<i>Name</i>	Wrapper Pattern
<i>Intent</i>	This pattern allows a legacy application to be incorporated into a MAS. The legacy code is extended with agent capabilities by agentifying it. Agentification may be a temporary solution until the legacy code can be upgraded or migrated to an agent implementation.
<i>Applicability</i>	If a legacy system provides functionality required by an agent system, can not easily be redeveloped as an agent and has a stable interface that can be wrapped, this pattern is appropriate. Here a legacy system is any non-agent application.
<i>Motivation</i> <i>Example</i>	This pattern enables the encapsulation of legacy code and enhancement of the legacy interface, to agentify the legacy. By introducing a layer of inter-agent communication functionality, the wrapper allows agents to communicate with the legacy using an ACL, and maps responses from the legacy back to ACL for the agents. This ensures that agent communication protocols are respected and the legacy code remains

	decoupled from the agents.
<i>Problem</i>	How can an agent application interact with a legacy system?
<i>Solution</i>	The agentified legacy should be capable of any of the supported interactions facilitated by the client ACL. The mapping of the interface to the legacy interface is system-specific.
<i>Participants</i>	This coordination pattern involves a wrapper, a number of client agents and generally only one legacy system, since the wrapper is domain or system-specific. The role of the wrapper is to interface the agents to the legacy system by agentifying the legacy.

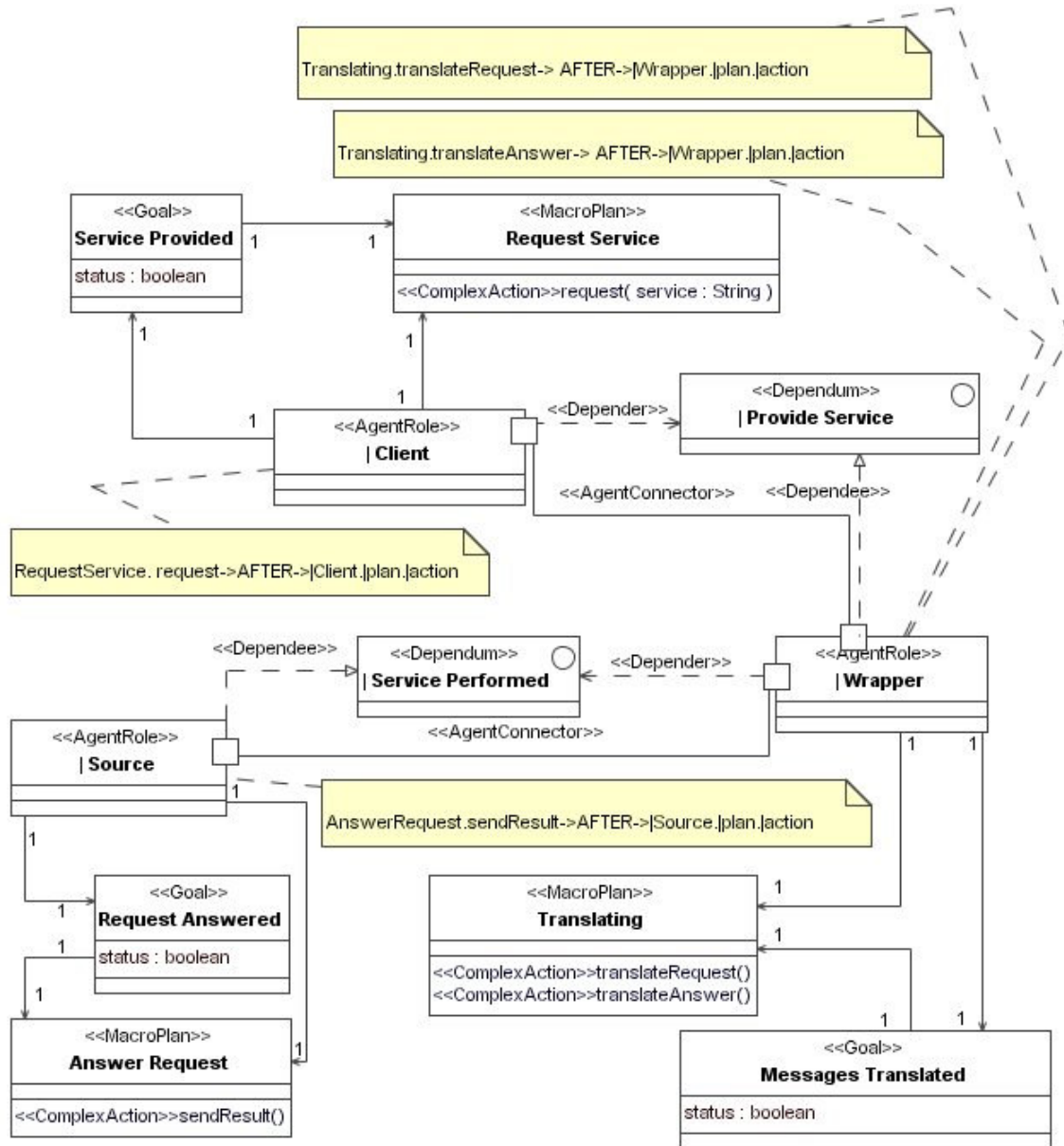


Figure 10 – Wrapper Structural Agent Pattern Specification

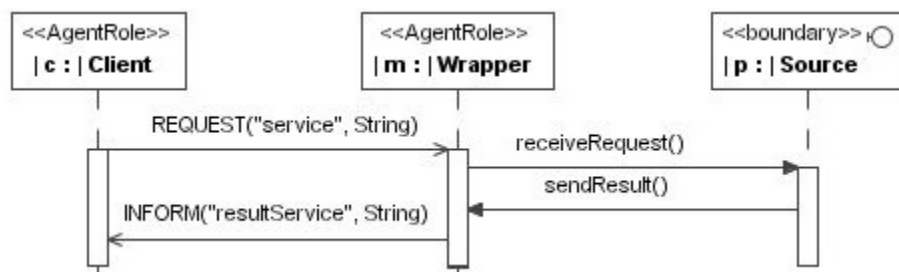


Figure 11 – Wrapper Communication Agent Pattern Specification

3.5 Broker

The Broker social pattern template is depicted in Table 11.6.

Table 6 – The Template for Broker Pattern

Element	Description
<i>Name</i>	Broker Pattern
<i>Intent</i>	The broker allows decoupling of the client and service-provider by accepting requests from a client, farming out the work to a willing and available service-provider, and returning results to the client. This allows for communication and location transparency for interoperating applications.
<i>Applicability</i>	If many clients and many service-providers exist in a particular application, this pattern is applicable. If distributed heterogeneous components are to be integrated, but their independence is to be maintained, this pattern is applicable. Distributed, heterogeneous components may not have been designed to interoperate: they may be dispersed over multiple platforms and may be implemented in different languages.
<i>Motivation Example</i>	In multi-agent systems involving numerous agents with a range of capabilities, it is not feasible for each client to hold capability models for each service-provider. This would result in a complete graph if agents held both roles of client and service-provider, and complex graphs for lesser cases, with consequent messaging overhead in maintaining the capability models. It is more efficient for a broker or a number of brokers to serve as go-betweens or matchmakers, maintaining capability models of service-providers and connecting clients to service-providers providing their needs.

<i>Problem</i>	How can clients request a service from unknown providers without interact directly with them?
<i>Solution</i>	The Broker pattern allows service providers to subscribe their services to the Broker agent, since the Broker agent plays the role of a yellow-page agent. The Client agent requests a service to the Broker agent. The Broker agent requests and receives services from service providers and then forwards the service to the Client agent.
<i>Participants</i>	This coordination pattern involves at least one broker, a number of clients/requesters and providers of services. The broker's role is that of arbiter and intermediary, accessing services of one agent to satisfy the requests of another. The clients may also be service providers, and the service providers may also be clients. The roles of the agents are established in the context of a particular conversation.

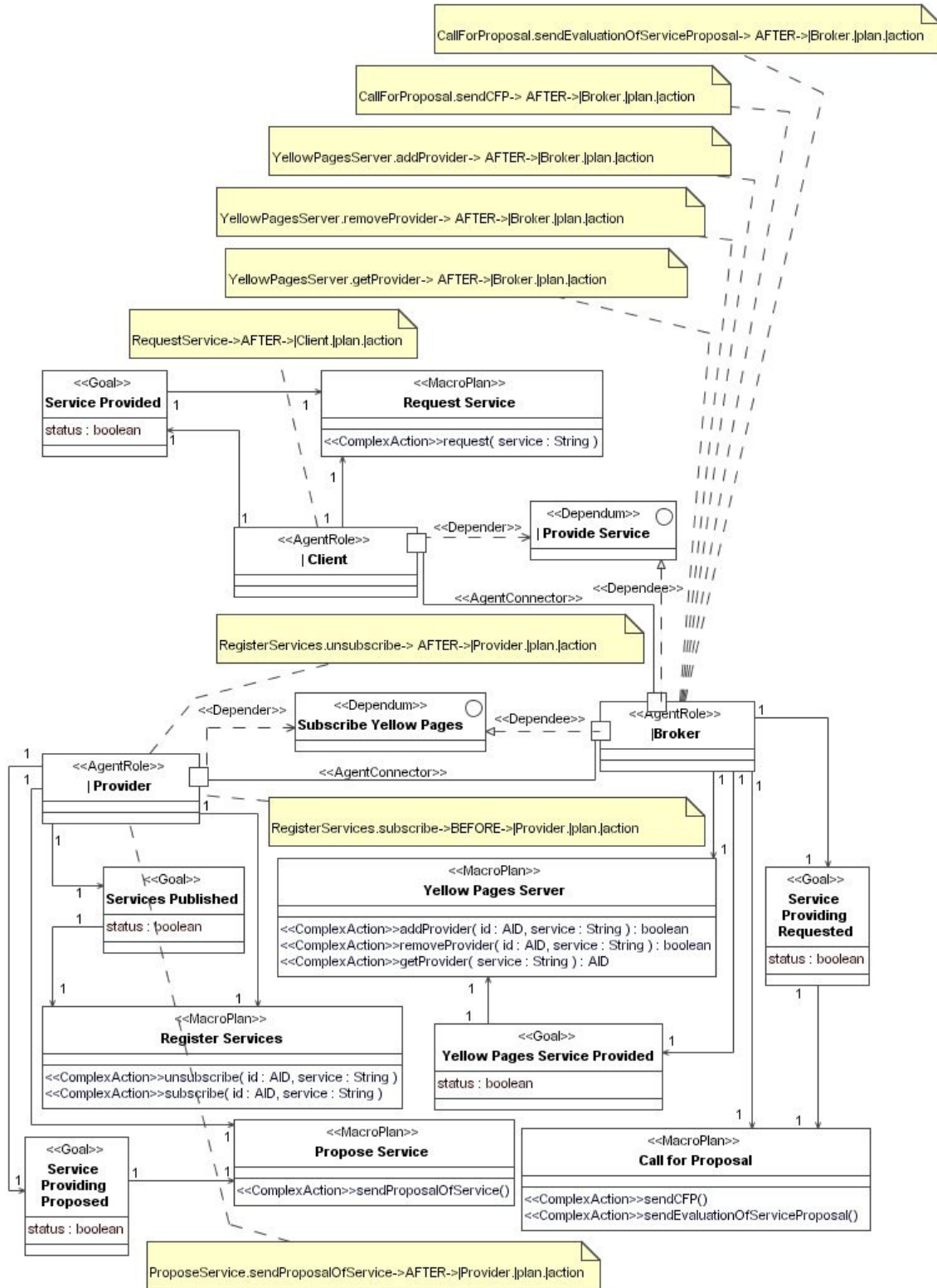


Figure 12 – Broker Structural Agent Pattern Specification

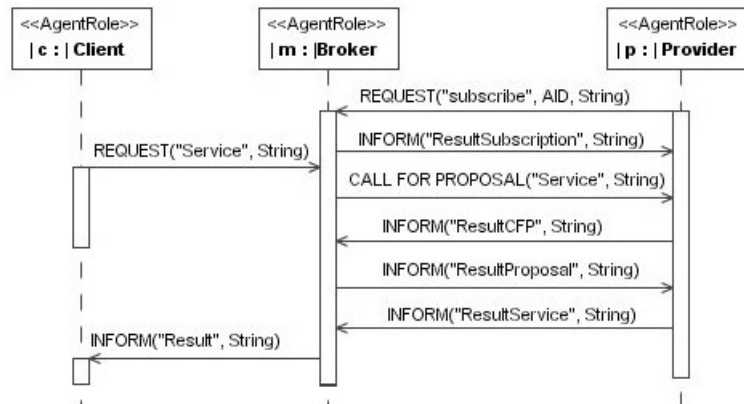
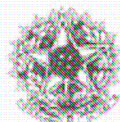


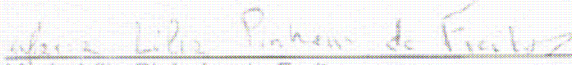
Figure 13 – Broker Communication Agent Pattern Specification



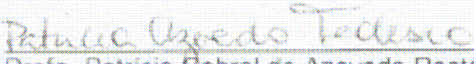
SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata de Defesa de Tese de Doutorado do
Centro de Informática da Universidade Federal
de Pernambuco, em 2 de abril de 2007.

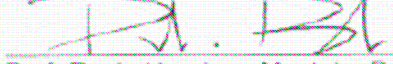
Ao segundo dia do mês de abril do ano dois mil e sete, no Centro de Informática da Universidade Federal de Pernambuco, às oito horas e trinta minutos, teve início a octogésima quinta defesa de tese do Doutorado em Ciência da Computação intitulada "Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case", elaborada sob a orientação do professor Jaelson Freire Brelaz de Castro, da candidata Carla Taciana Lima Lourenço Silva, a qual já havia preenchido as demais condições exigidas para a obtenção do grau de doutor. A Banca Examinadora composta pelos professores Patrícia Cabral de Azevedo Restelli Tedesco, Paulo Henrique Monteiro Borba, ambos pertencentes ao Centro de Informática desta Universidade, Fernanda Maria Ribeiro de Alencar, pertencente ao Departamento de Eletrônica e Sistemas desta Universidade, Paulo César Masiero, pertencente ao Instituto de Ciências Matemática e da Computação da Universidade de São Paulo e Jean-Pierre Briot, pertencente ao Laboratório de Informática da Universidade Paris 6, França, sendo a primeira presidente da Banca Examinadora, resolveu: **Aprovar por unanimidade e dar o prazo de noventa dias para a entrega da versão final do trabalho.** E para constar lavrei a presente ata que vai por mim assinada e pela Banca Examinadora. Recife, 2 de abril de 2007.



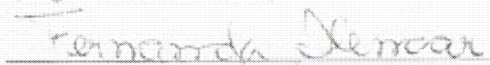
Mária Lília Pinheiro de Freitas
(secretária)



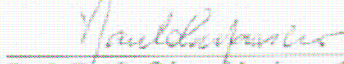
Profa. Patrícia Cabral de Azevedo Restelli Tedesco
(primeira examinadora)



Prof. Paulo Henrique Monteiro Borba
(segundo examinador)



Profa. Fernanda Maria Ribeiro de Alencar
(terceira examinadora)



Prof. Paulo César Masiero
(quarto examinador)



Prof. Jean-Pierre Briot
(quinto examinador)