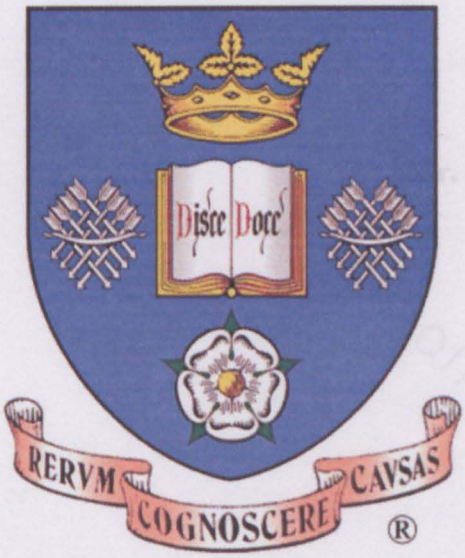
A SECURITY ORIENTED APPROACH IN THE DEVELOPMENT OF MULTIAGENT SYSTEMS:

APPLIED TO THE MANAGEMENT OF THE HEALTH AND SOCIAL CARE NEEDS OF OLDER PEOPLE IN ENGLAND

By Haralambos Mouratidis

Department of Computer Science

University of Sheffield, U.K.



January 2004

Dissertation submitted to the University of Sheffield for the degree of Doctor of Philosophy in Computer Science

DECLARATION

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or any other institution of learning.

Mr. Haralambos Mouratidis

January 22nd, 2004

ABSTRACT

Security can play an important role in the development of some multiagent systems. However, a careful analysis of software development processes indicates that the definition of security requirements is, usually, considered after the design of the system. This approach, usually, leads to problems, such as conflicts between security and functional requirements, which can translate into security vulnerabilities. As a result, the integration of security issues in agent oriented software engineering methodologies has been identified as an important issue. Nevertheless, developers of agent oriented software engineering methodologies have mainly neglected security engineering and in fact very little evidence has been reported on work that integrates security issues into the development stages of agent oriented software engineering methodologies.

This thesis advances the current state of the art in agent oriented software engineering in many ways. It identifies problems associated with the integration of security and software engineering and proposes a set of minimum requirements that a security oriented process should demonstrate. It extends the concepts and the development process of the Tropos methodology with respect to security to allow developers, even those with minimum security knowledge, to identify desired security requirements for their multiagent systems, reason about them, and as a result develop a system that satisfies its security requirements. In doing so, this research has developed (1) an analysis technique to enable developers to select amongst alternative architectural styles using as criteria the security requirements of the system, (2) a pattern language consisting of security patterns for multiagent systems, and (3) a scenario-based technique that allows developers to test the reaction of the system to potential attacks.

The applicability of the approach is demonstrated by employing it in the development of the electronic single assessment process (*eSAP*) system, a real-life case study that provided the initial motivation for this research.

DEDICATION

To my parents Vasili and Ourania Mouratidi

For giving me opportunities you never had...

For your love and support...

I thank you...

Στους γονείς μου Βασίλη και Ουρανία Μουρατίδη

Για τις ευκαιρίες που μου δώσατε, που εσείς δεν είχατε ποτέ...

Για την αγάπη και την υποστήριξή σας ...

Σας ευχαριστώ...

Χάρης

(January/Ιανουάριος 2004)

ACKNOWLEDGEMENTS

This thesis is the product of hard personal work. However, as any other research thesis, it would have never been completed without the help of some people. Although it is quite difficult to express in few paragraphs the gratitude I feel for these people, I will try my best.

First of all, I am grateful to my supervisors Gordon Manson and Ian Philp. Thanks to both for their patience (especially with my use of the English language), their guidance (Gordon on the computer science and Ian on the health and social care issues), support and for being available when I needed them. My thesis owes much to this unique combination of supervisors.

I would like to thank the financial support of the RANK Foundation and especially its representative Sq. Leader Larry Parsons for our cooperation throughout this research.

Thanks are also due to the members of my research panel Robert Gaizauskas, Gerald Lüttgen, and Anthony Simons for their useful comments and feedback throughout all the stages of my research project.

My special thanks goes to Paolo Giorgini for introducing me to the Tropos project and for the (many) discussions via e-mail or in person to clarify technical points.

Furthermore, thanks to the members of the Centre For Mobile Communications Research (C4MCR) for the long technical but also (more importantly) social discussions. Especially, thanks to Erika Sanchez for providing comments on my work and most importantly for being a good friend.

Thank you to all the academic and administrative staff of the Computer Science Department and the Sheffield Institute for Studies on Ageing (SISA), both at the University of Sheffield, for providing me with various types of help throughout my research.

Also thanks to Cos Tingle and Olivia Sarpong for proof reading my thesis, and to John Mylopoulos, Paolo Bresciani, James Odell, Michael Weiss, Michael

Schumacher, Paola Turci, for discussing various bits of my work and for providing valuable feedback.

Moreover, I would like to thank the various anonymous reviewers, of the papers that I have submitted to various journals and conferences, who have provided me with valuable feedback and interesting comments on how to improve my work.

The biggest part of everything I have achieved so far in my life, including this research, would not have been possible without the love and the support that I receive from my parents and my sister Chrysa. Μαμά, μπαμπά, Χρύσα σας ευχαριστώ για οτι έχετε κάνει, κάνετε και θα συνεχίσετε να κάνετε για μένα. Νιώθω τυχερός που έχω τέτοια οικογένεια.

I would like to thank my wife, Karina. Gracias por tu paciencia, apoyo y amor durante los muchos meses que he necesitado para terminar esta tesis y por proveer una dimensión diferente (placentera) a mi vida.

The chain of my gratitude would be definitely incomplete if I would forget to thank the Lord for keeping me in good health and providing me with the necessary courage to face the difficulties that happened during the time of this research.

Χάρης Μουρατίδης

PUBLICATIONS RESULTED FROM THIS RESEARCH

Excerpts of this thesis and in general work resulted from the presented research have been published in journals, conferences and workshops. Most notably the following:

- **H. Mouratidis**, I. Philp, G. Manson
"A Novel Agent-Based System to Support the Single Assessment Process for Older People", in *the Journal of Health Informatics* (9) 3, pp. 149-163, September 2003.

- **H. Mouratidis**, G. Manson, I. Philp
"Testing the suitability and the limitations of agent technology to support integrated assessment of health and social care needs of older people", in *Informatica Medica Slovenica* 8 (1), pp. 57-65, January 2003 (Invited paper. This is an extended version of the paper presented in CBMS 2002)

- **H. Mouratidis**, I. Philp, G. Manson
"Analysis and Design of eSAP: An Integrated Health and Social Care Information System", in the *Journal of Health Informatics* 9 (2), pp. 93-96, June 2003.

- **H. Mouratidis**, P. Giorgini, G. Manson
"Using Security Attack Scenarios to Analyse Security during Information Systems Design", (to appear) in the *Proceedings of the 6th International Conference on Enterprise Information Systems* (ICEIS-2004), Porto-Portugal, April 2004.

- **H. Mouratidis**, P. Giorgini, G. Manson
"An Ontology for Modelling Security: The Tropos Approach", in the *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems* (KES 2003), Invited Session on Ontology and Multi-Agent Systems Design (OMASD'03), Oxford-England, September 2003.

- **H. Mouratidis**, P. Giorgini, G. Manson
"Modelling Secure Multiagent Systems", in the *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne-Australia, ACM press, July 2003.

- P. Bresciani, P. Giorgini, **H. Mouratidis**, G. Manson
"Multi-Agent Systems and Security Requirements Analysis", book chapter in *Advances in Software Engineering for Multi-Agent Systems*, Carlos Lucena, Alessandro Garcia, Alexander Romanovsky, Jaelson Castro, Paulo Alencar (editors), Lecture Notes in Artificial Intelligence, Springer-Verlag, 2003 (in press). (Revised and extended version of the paper appeared in SELMAS '03)

- **H. Mouratidis**, P. Giorgini, M. Schumacher
"Security Patterns for Agent Systems", in the *Proceedings of the 8th European Conference on Pattern Languages of Programs* (EuroPLoP), Isree-Germany, June 2003.

- **H. Mouratidis**, P. Giorgini, G. Manson
"Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems", in the *Proceedings of the 15th Conference on Advance Information Systems* (CAiSE 2003), Velden-Austria, June 2003.

- **H. Mouratidis**, P. Giorgini, G. Manson, A. Gani
"Analysing Security Requirements of Information Systems Using Tropos", in the *Proceedings of the 5th International Conference on Enterprise Information Systems* (ICEIS-2003), Angers-France, April 2003.

- **H. Mouratidis**, P. Giorgini, M. Weiss
"Integrating Patterns and Agent Oriented Methodologies to Provide Better Solutions for the Development of Secure Agent-Based Systems", in the *6th ChiliPLoP Annual Conference*, Hot Topic: Expressiveness of Pattern Languages, Arizona-USA, March 2003.

- **H. Mouratidis**, P. Giorgini, G. Manson, I. Philp
"A Natural Extension of the Tropos Methodology for Modelling Security", in the *Proceedings of the Agent Oriented Methodologies Workshop* (at the OOPLSA 2002), Seattle-USA, November 2002.

- **H. Mouratidis**, G. Manson, I. Philp, P. Giorgini
"Modelling an agent-based integrated health and social care information system for older people", in the *Proceedings of the International Workshop on Agents Applied in Health Care* (at the 15th European Conference on Artificial Intelligence), Lyon-France, July 2002.

- **H. Mouratidis**, P. Giorgini, G. Manson, I. Philp
"Using Tropos methodology to Model an Integrated Health Assessment System", in the *Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS-2002), Toronto-Ontario, May 2002.

- **H. Mouratidis**, G. Manson, I. Philp
"Testing the suitability and the limitations of agent technology to support integrated assessment of health and social care needs of older people", in the *Proceedings of the 15th IEEE Symposium on Computer Based Medical Systems* (CBMS-2002), Maribor-Slovenia, June 2002 (Best Student Paper Award).

- **H. Mouratidis**
"An Agent-Oriented Methodology for the Development of Secure Agent-Based Systems", in the *Proceedings of the PREP 2003*, Exeter-England, April 2003.

- **H. Mouratidis**, I. Philp, G. Manson
"Towards a mature and complete agent-oriented software engineering methodology", in the *Proceedings of the PREP 2002*, Nottingham-England, April 2002.

Moreover, two journal papers (one based on the analysis and design of the electronic single assessment process system and the other on chapters 4 and 5 are in preparation.

TABLE OF CONTENTS

Declaration i

Abstract ii

Dedication iii

Acknowledgements iv

Publications resulted from this research vi

Table Of contents ix

List Of figures xiv

List of tables xvii

CHAPTER 1 Introduction 1

1.1 Motivation of this research 1

1.2 The problem 2

1.3 Related work: existing approaches and their limitations 4

1.4 Research aims and approach 8

1.5 Structure of the thesis 9

CHAPTER 2 Agent oriented software engineering and security modelling 11

2.1 A brief review of basic concepts in software engineering 11

2.1.1 Requirements engineering 13

2.1.2 Design stage 15

2.1.3 Development methodologies 15

2.2 Agent oriented software engineering 17

2.2.1 Agents and multiagent systems 17

2.2.2 Defining the agent oriented software engineering paradigm 19

2.2.2.1 Agents as a modelling construct 20
2.2.2.2 The arguments for the use of agent oriented software engineering paradigm 21
2.2.3 Agent oriented software engineering methodologies 22
2.3 Security modelling 26
2.3.1 Basic security concepts and ideas 26
2.3.2 Security in software engineering 28
2.3.2.1 Problems of modelling security during the development of a system 29
2.3.2.2 Requirements of a security-oriented approach 30
2.4 Agent oriented software engineering and security engineering 31
2.4.1.1 Identification of a suitable methodology 32
2.5 Summary 37
CHAPTER 3 An overview of the Tropos methodology 38
3.1 An introduction to the Tropos methodology 38
3.2 A review of Tropos concepts and notations 40
3.3 The stages of the Tropos methodology 42
3.4 The modelling language of Tropos 43
3.5 Modelling activities in Tropos 44
3.6 A set of transformations 52
3.7 An example of using Tropos 53
3.7.1 Early requirements analysis stage 53
3.7.2 Late requirements analysis stage 58
3.7.3 Architectural design stage 60
3.7.4 Detailed design stage 65
3.8 Limitations of Tropos with respect to security modelling 67
3.8.1 Limitations on the concepts of the methodology 68

3.8.2 Limitations on the Tropos' process of modelling security 69
3.8.3 Discussion with respect to the limitations 71
3.9 Summary 72
CHAPTER 4 Secure concepts and modelling activities 73
4.1 Integrating security in the Tropos methodology 73
4.2 The secure concepts 74
4.2.1 Constraint and security constraint 74
4.2.2 Secure dependency 78
4.2.3 Secure entities 80
4.2.4 Secure capability 81
4.3 Modelling activities 81
4.3.1 Security reference diagram modelling 82
4.3.1.1 Nodes of the security reference diagram 83
4.3.1.2 Links of the security reference diagram 85
4.3.1.3 An example of a security reference diagram 85
4.3.1.4 A transformation system for the construction of the security reference diagram 86
4.3.1.5 Algorithm for the construction of the security reference diagram 89
4.3.2 Security constraint modelling 91
4.3.2.1 Security constraint delegation and assignment 92
4.3.2.2 Security constraint analysis 93
4.3.3 Secure entities modelling 95
4.3.4 Secure capability modelling 95
4.4 Summary 95
CHAPTER 5 A security-oriented process 97
5.1 Identifying the security requirements of the system 97

5.2 Selecting amongst alternative architectural styles 98
5.3 Towards a design that satisfies the security requirements 100
5.3.1 Security patterns for agent systems 101
5.3.2 The pattern language 102
5.3.2.1 A description of the patterns 105
5.3.2.2 An example of using the pattern language 113
5.4 Attack testing of the multiagent system under development 114
5.4.1 Scenario creation 117
5.4.1.1 Identify the intentions of possible attackers 117
5.4.1.2 Identify possible countermeasures 118
5.4.2 Scenario validation 119
5.4.3 Testing and redefinition of the system 120
5.5 Checking the consistency of the security process 121
5.5.1 Consistency rules 122
5.6 Refining the Tropos stages to include the security-oriented process ... 123
5.7 Summary 124
CHAPTER 6 Applying the extensions: the eSAP case study 126
6.1 The single assessment process and the motivation behind the electronic single assessment process 126
6.2 A typical scenario 127
6.3 developing the eSAP 130
6.3.1 Early requirements analysis 130
6.3.2 Late requirements analysis 140
6.3.3 Architectural design 143
6.3.3.1 Interception Scenario 154
6.3.3.2 Modification Scenario 157
6.3.3.3 Interruption Scenario 160

6.3.3.4 Discussion regarding the security attack scenarios 161
6.3.4 Detailed design 163
6.4 Discussion regarding the security extensions 167
6.4.1 How the proposed security-oriented approach helps towards the development of secure multiagent systems 167
6.4.2 The key features of the proposed approach 169
6.5 Summary 170
CHAPTER 7 Conclusions 171
7.1 Does the proposed approach satisfy the requirements set in section 2.3.2.2? 171
7.2 Discussion on how the research objectives were met 173
7.3 Integration to other methodologies 178
7.4 Research contributions 180
7.5 Significance of this research 181
7.6 Directions for future work 183
7.7 Summary 186
Appendix A: Consistency rules 187
Appendix B: Supported material for chapter 6 190
References 205

LIST OF FIGURES

Figure 3-1: Graphical representation of the Tropos concepts 42
Figure 3-2: An example of an actor diagram 47
Figure 3-3: An example of a goal diagram 48
Figure 3-4: An example of a capability diagram 50
Figure 3-5: An example of a plan diagram 51
Figure 3-6: An example of an agent interaction diagram 52
Figure 3-7: The actor diagram for the given example 54
Figure 3-8: Part of the goal diagram for the Older Person 55
Figure 3-9: Part of the goal diagram for the Department of Health 57
Figure 3-10: Part of the goal diagram for the eSAP 58
Figure 3-11: Partial decomposition of the eSAP actor 61
Figure 3-12: Part of the extended actor diagram with respect to the *Obtain Information about Care Plan* task of the Older Person 63
Figure 3-13: Extended diagram with respect to the Authorisation Manager 64
Figure 3-14: Capability diagram for the authorisation status capability 65
Figure 3-15: Plan diagram for the evaluate authorisation status plan 66
Figure 3-16: Example of an agent interaction diagram 67
Figure 4-1: UML metamodel for the concept of constraint 76
Figure 4-2: Graphical representation of a constraint and a security constraint 78
Figure 4-3: Graphical representation of secure dependencies 79
Figure 4-4: Graphical representation of secure entities 81
Figure 4-5: Graphical representation of a capability and a secure capability 81
Figure 4-6: Graphical representation of nodes used in the security reference diagram 84
Figure 4-7: Positive and Negative Contribution links 85
Figure 4-8: Example of a security reference diagram 86
Figure 4-9: Example of a security constraint delegation 92

Figure 4-10: Example of a security constraint assignment 93
Figure 4-11: Example of security constraint decomposition 94
Figure 4-12: Example of secure goal introduction 94
Figure 5-1: An example of selecting amongst architectural styles 100
Figure 5-2: Relationships between the patterns of the language and other existing patterns 104
Figure 5-3: The AGENCY GUARD dependencies 106
Figure 5-4: The AGENT AUTHENTICATOR dependencies 109
Figure 5-5: The SANDBOX dependencies 111
Figure 5-6: The ACCESS CONTROLLER dependencies 113
Figure 5-7: Example of a goal diagram analysing the intentions of an attacker 118
Figure 5-8: An example of a security attack scenario 119
Figure 6-1: Security reference diagram 131
Figure 6-2: The actor diagram 135
Figure 6-3: Goal diagram for the Nurse actor 137
Figure 6-4: Goal diagram of the Older Person actor 138
Figure 6-5: Refined actor diagram 139
Figure 6-6: Actor diagram including the eSAP actor 141
Figure 6-7: Goal diagram for the eSAP actor 142
Figure 6-8: Client/Server versus Mobile Agents architectural styles 144
Figure 6-9: Decomposing the eSAP system 146
Figure 6-10: Using the AGENCY GUARD, the AGENT AUTHENTICATOR and the ACCESS CONTROLLER patterns in the development of the eSAP 148
Figure 6-11: Decomposition of the authorisation and integrity managers 149
Figure 6-12: Extended diagram for the eSAP 150
Figure 6-13: Extended actor diagram with respect to the Assessment Evaluator 151
Figure 6-14: Interception attacks scenario 154
Figure 6-15: Modification attacks scenario 157
Figure 6-16: Interruption attacks scenario 160
Figure 6-17: Partial Class Diagram with respect to the Meeting Scheduler 164
Figure 6-18: Capability diagram for the Receive Care Plan Request capability of the Care Plan Broker agent 164

Figure 6-19: Plan diagram for the evaluate care plan request plan 166
Figure 6-20: Interaction diagram for the Social Worker system access clearance .. 167
Figure 7-1: A global architecture for a Tropos tool 185

LIST OF TABLES

Table 2-1:Evaluation of the methodologies 35
Table 3-1: Actors and their capabilities with respect to Figure 3-13 64
Table 5-1: Example of consistency rules 123
Table 6-1: Actors and their capabilities with respect to the extended diagram of Figure 6-13 152
Table 6-2: Agent types and their capabilities 153
Table 6-3: The agents of the eSAP System 162

CHAPTER 1 INTRODUCTION

This thesis reports on novel work, in the area of agent oriented software engineering, which integrates security issues in agent oriented development. The main novelty lies in the fact that the same concepts and notations are used throughout the development process and a clear and well-structured security-related process is provided (applicable even by less security-oriented developers) to consider security issues during the development of multiagent systems. In particular, this thesis presents security-related concepts, notations, models and procedures and their integration within the development stages of the Tropos methodology [Giu02], a widely known agent oriented software engineering methodology. Additionally, this thesis describes how the extended, with respect to security, Tropos methodology can be applied in the development of a real-life agent oriented information system for the assessment of the health and social care needs of older people in England.

This introductory chapter forms an overview of the thesis. Section 1.1 describes the main motivation behind the presented research, and section 1.2 presents the problem that this research addresses. Section 1.3 provides an overview of the state of the art by discussing work related to this research and section 1.4 outlines the research aims and the approach followed in order to successfully complete the identified aims. Moreover, section 1.5 presents the structure of the rest of the thesis.

1.1 MOTIVATION OF THIS RESEARCH

This research started as an effort to develop an information system to deliver the single assessment process, an integrated assessment of health and social care needs of older people in England [Phil97, Doh03]. Such a system is considered very important by the English Department of Health since it has the potential to improve efficiency and effectiveness in the collection and sharing of assessment information regarding older people.

Towards the development of the electronic single assessment process (eSAP) system, this research project identified agent oriented software engineering [Wool01]

as a suitable paradigm. This is mainly due to the fact that the level of abstraction that agent oriented software engineering brings in the development of complex computerised systems, such as the electronic single assessment process system, helps in better mutual understanding between system developers (computer scientists) and system users (health and social care professionals, and older person in the case of eSAP). This is because system developers can better explain the functionalities of the system, by decomposing it to smaller autonomous entities (agents) that possess characteristics similar to humans, such as mobility and the ability to communicate, and the system users can use the concept of an agent to describe more precisely the needs of the system.

However, in trying to employ agent oriented software engineering in the development of the electronic single assessment process system very little help was found. Current agent oriented software engineering methodologies are neither complete nor adequate for the development of the electronic single assessment process system. The main deficiency identified was the lack of models and a structured process, which uses the same concepts and notations, to model security issues throughout the whole development lifecycle.

Having identified the fundamental problem, the motivation of this research was directed towards its solution.

1.2 THE PROBLEM

In software engineering, the common approach towards the inclusion of security within a software system is to identify security requirements after the definition of a system [Dev00, Lod02]. This typically means that security enforcement mechanisms have to be fitted into a pre-existing design. This approach leads to serious design challenges that usually translate into the emergence of computer systems afflicted with security vulnerabilities [And01, Sta99]. However, security is of particular importance to multiagent systems as these are, by design, built as open systems and interactions will take place between agents of different systems that are not known to the developers during design. As a result, security is considered one of the main issues to be dealt for agent technology to be widely used outside the research community [Jan00, Mou03].

However, research efforts so far have been mainly focused on the solution of individual security problems of multiagent systems, such as attacks from an agent to another agent, attacks from a platform to an agent, and attacks from an agent to a platform [Jan99]. Developers of agent oriented methodologies have mainly neglected security and although the agent oriented software engineering is progressing rapidly and many agent oriented methodologies [Eva01, Giu02, Igl97, Igl99, Wood01, Wool99] have been developed during the last few years, agent oriented software engineering practises and methodologies do not meet the needs for resolving the security related problems, and fail to provide evidence of successfully integrating security concerns. As a result, multiagent system developers find no help when considering security during the development of multiagent systems.

The problem is stated as follows:

The lack of an agent oriented software engineering methodology to assist developers in considering security issues during the development of multiagent systems throughout all the development stages.

Observations related to this problem have been presented in the literature. Tryfonas et al. [Try97] note that existent methodologies for information system development fail to include specialised handling of the security requirements, and they do not create a control environment early in the development process. Fischer et al. [Fis02] indicate that little research has been carried out to integrate individual security techniques into a global methodology for agent technologies and multiagent systems.

Moreover, Devanbu and Stubblebine [Dev00] argue that security should inform every phase of software development, from requirements engineering to design, implementation, testing and deployment. They point out that a major challenge is to unify security and systems engineering in order to deploy available resources and build the right combination of customer features and security measures.

On the other hand, factors such as the involvement of non-security experts in the development of multiagent systems, which do require knowledge of security, and the difficulty of moving from a set of security requirements to a design that satisfies these requirements, contribute to the difficulty of the above mentioned problem. Therefore a solution to this problem should allow even non-security specialists to reason about security when developing a multiagent system, and also it should use

the same concepts and notations throughout the development process in order to limit possible inconsistency that appear due to the translation of concepts when the software process moves from one development stage to another. By considering the above mentioned observations and factors, the problem can be re-stated as follows:

The lack of an agent oriented software engineering methodology to assist (even non-security oriented) developers in considering security issues during the development of multiagent systems using the same concepts and notations throughout all the development stages.

1.3 RELATED WORK: EXISTING APPROACHES AND THEIR LIMITATIONS

This section describes existing state of the art approaches and indicates why these approaches are limited and do not adequately solve the problem.

As mentioned above, current agent oriented methodologies do not meet the needs for resolving the security related problems, and fail to provide evidence of integrating successfully security concerns throughout the whole range of the development process. Nevertheless, recently, some work has been initiated towards the solution of the problem.

Liu et al. [Liu02] have presented work to identify security requirements during the development of multiagent systems. In this work, security requirements are analysed as relationships amongst strategic actors, such as users, stakeholders and potential attackers. Liu proposes three different kinds of analysis techniques: agent oriented, goal oriented and scenario based analysis. Agent oriented analysis is used to model potential threats and security measures, whereas goal oriented analysis is employed for the development of a catalogue to help towards the identification of the different security relationships on the system. Finally, the scenario based analysis is considered an elaboration of the other two kinds of analysis.

In addition, Yu and Cysneiros [Yu02] provide an approach to model and reason about non-functional requirements (with emphasis on privacy and security). They are using the concept of a soft-goal to assess different design alternatives, and how each of these alternatives would contribute positively or negatively in achieving the soft-goal.

Both of these works are mainly focused only in the requirements analysis area and not in the whole development process. In addition, both Liu and Yu employ the concept of a soft-goal to help them in their analysis. Although soft-goals provide a good idea regarding the security of the system during the requirements analysis, they do not provide enough detail when considering security in the other stages of the development process. Therefore, as it has been argued in the literature [Mou02] (and presented in section 3.8.1), the concept of a soft-goal does not adequately model security issues throughout the development process.

Moreover, Huget [Hug02] proposes a new agent oriented methodology, called Nemo and claims that it tackles security. In his approach, security is not considered as a specific model but it is included within the other models of the methodology. Nemo is a new methodology and as a result it has not been extensively presented on the literature. However, from the point of view of this research, the methodology tackles security quite superficial and as the developer states *"particularly, security has to be intertwined more deeply within models"* [Hug02]. Therefore, more evidence will be required to satisfy the claim of the developer that the methodology tackles security.

The above presented attempts are focused on the integration of security issues within the agent oriented software engineering paradigm. Most of the attempts, however, to integrate security and software engineering come from close disciplinary areas such as requirements engineering, object oriented software engineering and patterns. In the current state of the art, security properties are mainly supported by a qualitative reasoning rather than a formal reasoning within the requirements engineering process.

Chung applies a process-oriented approach [Chu95] to represent security requirements as potentially conflicting or harmonious goals and using them during the development of software systems. The proposed framework, which is called the NFR (Non-Functional Requirements) framework, represents and uses security requirements as a class of non-functional requirements and it allows developers to consider design decisions and relate these decisions to the represented non-functional requirements.

Rohrig [Roh02] proposes an approach to re-use existing business process descriptions for the analysis of security requirements and the derivation of necessary security measures. The proposed approach consists of four main steps. During the first step, the general security objectives of the business process are defined, whereas during the second step the security objectives of all the constructs, such as actors and artefacts, are examined. The third step examines whether these specifications are consistent and during the fourth step a list of necessary security measures for each process component is generated.

In addition, Jurgens proposes UMLsec [Jur01, Jur02], an extension of the Unified Modelling Language (UML), to include modelling of security related features, such as confidentiality and access control. In his work, Jurgens uses four different UML diagrams; class diagrams to ensure that exchange of data obeys security levels, state-chart diagrams to prevent indirect information flow from high to low values within an object, interaction diagrams to ensure correctness of security critical interactions between objects and deployment diagrams to ensure that security requirements on communication are met by the physical layer.

Lodderstedt et al. [Lod02] also extend UML to model security. In their work, they present a security modelling language called SecureUML [Lod02]. They describe how UML can be used to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures.

McDermott and Fox [Mcd99] adapt use cases to capture and analyse security requirements, and they call the adaption an abuse case model. An abuse case is defined as a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders of the system.

Sindre and Opdahl [Sin00] define the concept of a misuse case, the inverse of a use case, which describes a function that the system should not allow. They also define the concept of a mis-actor as someone who intentionally or accidentally initiates a misuse case and whom the system should not support in doing so. In their approach security is considered by analysing security related misuse cases.

Scheneir [Sch00] describes attack trees as a useful way to identify and organise different attacks in an information system. According to Scheneir, attack trees represent a set of intrusion scenarios and allow the refinement of attacks to a level of detail chosen by the developers. The root of the tree represents the compromise of a function of the system, whereas the nodes indicate a sequence of attack steps, represented as an AND-Decomposition, or alternative ways of executing the attack, represented as an OR-Decomposition.

Schumacher and Roedig [Schu01] apply the pattern approach to the security problem by proposing a set of patterns, called security patterns, which contribute to the overall process of security engineering. As they argue [Schu01], security patterns help security novices to act as security experts, and allow security problems to be solved in a structured way.

The concept of obstacle is used in the KAOS framework [Dar91] to capture undesired properties of the system, and define and relate security requirements to other system requirements. In this work, two set of techniques, based on a temporal logic formalisation, are employed to reason about obstacles to the satisfaction of goals, requirements, and assumptions elaborated in the requirements engineering process.

These (above-mentioned) approaches provide a first step towards the integration of security and software engineering and have been found helpful in modelling security requirements. However, they only guide the way security can be handled within a certain stage of the software development process. For example, McDermott and Fox's approach is used only during the requirements analysis, whereas Jurgen's analysis take place in a fairly low level and it is suited to a more operational analysis. In other words, Jurgen's approach is only applicable during the design stage.

Differently than them, this research proposes an approach that covers the whole development process using the same concepts and notations. As argued in this thesis, considering security issues throughout the development process by using the same concepts and notations is very important when developing multiagent systems with security on mind. By considering security only in certain stages of the development process, more likely, security needs will conflict with functional requirements of the system. On the other hand, considering security throughout all the development

process helps to limit the cases of conflict, by identifying them very early in the system development, and find ways to overcome them.

Moreover, some of the above mentioned approaches only deal with specific security issues. For example, SecureUML is focused more in access control policies and how these policies can be integrated into a model-driven software development process. Although such an analysis is important, it is very specific and it is applicable only on the design stage of the modelling process. In contrast, the approach presented in this thesis considers the whole range of security issues, from access control to authentication and integrity.

In addition to the above approaches, existing formal methods [Ban89, Rya00] support the verification of a security protocol, which has already been specified [Mea94]. However, such approaches are only applicable by security specialists and cannot be easily applied by software developers. On the other hand, the approach presented in this thesis uses concepts and notations derived mainly from the (agent oriented) software engineering area and as a result can be applied by software developers with minimum knowledge of security engineering.

1.4 RESEARCH AIMS AND APPROACH

The main aim of this research is to provide an answer to the problem mentioned in section 1.2. In other words, ***this research aims to provide an agent oriented software engineering methodology to assist (even non-security oriented) developers in considering security issues during the development of multiagent systems using the same concepts and notations throughout all the development stages.***

To accomplish this aim the following objectives have been identified:

- Identify problems of integrating security and systems engineering and provide a set of minimum requirements necessary for a security oriented process.
- Extend the concepts and notations of an existing agent oriented software engineering methodology with respect to security modelling.
- Develop a clear, well guided process of integrating security and systems engineering throughout the software development process of multiagent systems, using the same concepts and notations throughout the process.

- Integrate the security oriented process within the methodology's development stages.
- Evaluate the proposed solution by applying it for the development of the electronic single assessment process system.

As indicated by the above objectives, instead of developing a new methodology, this research project extends an existing agent oriented software engineering methodology. Mainly this decision took place in order to take advantage of existing work in agent oriented methodologies and focus on the integration of security and systems engineering rather than the development of a new methodology. To this extend, several agent oriented software engineering methodologies were reviewed and the Tropos agent oriented methodology was identified as the most suitable for the purposes of this project. Then the limitations of Tropos with respect to security were identified and new security related concepts were introduced to the methodology. Also, existing concepts were identified with security in mind and a security oriented process was developed and integrated within the development stages of the Tropos methodology.

To evaluate the proposed solution, the extended Tropos methodology has been applied in the development of a real life health and social care information system that provided the initial motivation for this research.

1.5 STRUCTURE OF THE THESIS

The structure of the rest of this thesis is as follows.

Chapter 2 introduces some basic software engineering concepts, such as requirements engineering and development methodologies, and it describes the agent oriented software engineering paradigm. Furthermore, the problems of modelling security issues during the development lifecycle are outlined, and a set of requirements, developed by this research project, necessary for a security-oriented process is presented. This chapter also identifies the methodology to be used by this research project for the integration of the proposed security-oriented process.

Chapter 3 provides a necessary overview of the Tropos methodology. The basic advantages and the key features of the Tropos methodology are presented and the methodology's main concepts, notations and development stages are introduced. To

facilitate better understanding of the methodology, an example is used. In addition, a critical discussion of the methodology's limitations with respect to security modelling is presented.

Chapter 4 describes security-oriented extensions to the concepts and the modelling activities of the Tropos methodology. Thus, this chapter discusses how this research approached the issue of integrating security in the Tropos methodology and it then describes the newly introduced and the extended concepts as well as the modelling activities with respect to the security modelling.

Chapter 5 describes the security-oriented process proposed by this research. This process includes the identification of security requirements of a multiagent system, the selection amongst alternative architectural styles for the system-to-be according to the identified security requirements, the development of a design that satisfies the security requirements of the system, and the attack testing of the multiagent system under development. Moreover, the chapter describes how the proposed process has been integrated within the Tropos development stages.

Chapter 6 describes how the security-aware Tropos methodology can be employed in the development of the electronic single assessment process (eSAP) system, a real-life case study that provided the initial motivation for this research. An introduction to the single assessment process is provided and the motivations behind the development of the electronic single assessment process system are outlined. Moreover, the chapter describes a typical scenario regarding the single assessment process and a description of developing the electronic single assessment process with the extended security-aware Tropos methodology. In addition, the chapter provides a critical discussion/evaluation regarding the proposed security-oriented approach.

Chapter 7 concludes this thesis. It discusses how the presented approach successfully satisfies the requirements, regarding a security oriented approach, set on section 2.3.2.2 and also how this research project met its objectives. Moreover, it discusses the contributions and the significance of this research and it describes directions for future work.

CHAPTER 2 AGENT ORIENTED SOFTWARE ENGINEERING AND SECURITY MODELLING

The previous chapter formed an introduction to this thesis. This chapter aims to provide readers with the necessary background to better understand the rest of this thesis. The problems of modelling security issues during the development lifecycle are outlined, and a set of requirements, developed by this research project, necessary for a security-oriented process is presented. This chapter also identifies the methodology used by this research for the integration of the proposed security-oriented process.

The chapter is divided into four main sections. Section 2.1 introduces readers to some basic concepts of software engineering, such as requirements engineering, the design development stage and software development methodologies, and it provides enough background to proceed to section 2.2 of the chapter, in which the concepts of agent and multiagent systems are described and the agent oriented software engineering paradigm is defined. A discussion regarding agent oriented software engineering methodologies concludes this section. Section 2.3 introduces security modelling and it identifies the problems of modelling security during the development of a system. In addition, a minimum set of requirements of a security oriented approach is outlined. Section 2.4 discusses the suitability of the agent oriented software engineering paradigm for the integration of security modelling in software engineering, and it identifies a suitable agent oriented software engineering methodology for the integration of security modelling during the development stages of a multiagent system. Finally, section 2.5 summarises the chapter.

2.1 A BRIEF REVIEW OF BASIC CONCEPTS IN SOFTWARE ENGINEERING

Trying to explicitly and accurately define something as wide and dynamic as

software engineering is a very difficult task. Therefore, there is a tendency from researchers to keep inventing new definitions according to a particular research project. As a result of this, various different definitions regarding software engineering appear on texts [Som01, Mac90, Vli93]. These definitions often use different words and different ideas to describe software engineering and range from very simple ones, such as *software engineering is what software engineers do* (a phrase that came up some times in discussions the author had with different people about software engineering), to very complicated ones.

The rest of this section presents some of the existing definitions and concludes with a definition of software engineering, within the context of this project, that uses existing terminology and captures the essentials of the presented definitions.

An early definition about software engineering was given at a NATO conference held at 1968. According to the final report of this conference [Nau68], "*Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*".

Extending this definition, Macro and Buxton [Mac90] claim "*software engineering is the establishment and use of sound engineering principles and good management practice, and the evolution of applicable tools and methods and their use as appropriate, in order to obtain – within known and adequate resource provisions – software that is of high quality in an explicitly defined sense*". A definition closely related with the one presented by Macro and Buxton is presented by Fairley [Fai85]. According to this, software engineering is the technological and managerial discipline concerned with the systematic production and maintenance of software products that are developed and modified on time and within cost estimates [Fai85].

Similarly to the above definitions, the IEEE Standard Glossary of Software Engineering Terminology [IEEE90] defines software engineering as "*the application of systematic, disciplined, quantifiable approach to the development operation and maintenance of software; that is the application of engineering to software*".

On the other hand, Morven Gentleman argues that software engineering is the use of methodologies, tools, and techniques to resolve the practical problems that arise in the construction, deployment, support and evolution of software

[http://wwwsel.iit.nrc.ca/sedefn/SEdefn.html], whereas David Fisher defines software engineering as the study of systematic and effective processes and technologies for supporting software development and maintenance activities [http://edlab-www.cs.umass.edu/cs320/lectures/1b-intro.PDF].

In this research, ***software engineering is defined as an engineering approach to the software systems development that provides methodologies, tools and techniques to help software system developers in the analysis, design, implementation and testing of software systems.***

2.1.1 Requirements engineering

An early step of the software engineering process is the requirements analysis stage. This section aims to describe requirements engineering, a term that covers all the activities involved during the requirements analysis stage, and also to point out why requirements engineering is an important part of the software engineering development process.

Requirements are defined during the early stages of a system development as a specification of what should be implemented [Som99]. Usually requirements are divided into two main categories, functional and non-functional requirements [Som01]. Functional requirements describe what the system should do, whereas non-functional requirements introduce quality characteristics and represent constraints under which the system should operate. Non-functional requirements usually include performance, accuracy, user-friendliness, availability, and security.

To help developers to correctly acquire requirements, a relatively new term that covers all of the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system has been invented: Requirements Engineering. According to Axel van Lamsweerde [Lam00] "*Requirements engineering is concerned with the identification of the goals to be achieved by the envisioned system, the operationalisation of such goals into services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software*".

Eric Yu [Yu97] argues that requirements engineering involves two main stages, an early requirements analysis and a late requirements analysis. Early

requirements analysis considers how the system would meet the organisational goals, why the system is needed, what the implications of the alternatives are for the various stakeholders[1] and how the stakeholders' interests and concerns might be addressed. Therefore, the emphasis during the early requirements analysis is on understanding the *whys* rather the *what* the system should do.

The *what* the system should do is considered during the late requirements analysis. This involves the precise and detailed specification of what a system should do. In other words, during the late requirements analysis a detailed description is provided on how the system should behave and/or what are its properties.

This research adopts the above views of both Axel van Lamsweerde, with respect to requirements engineering, and Yu with respect to the differentiation between early and late requirements analysis. Moreover, throughout this research, requirements engineering is treated as an important and crucial stage for the successful development of a system. This is mainly because when mistakes take place during this stage, these mistakes are usually propagated in the following stages of the development. This argument is supported by an estimation presented by Boehm, [Boe81] and according to which, the late correction of requirements errors could cost up to 200 times as much as correction during the requirements analysis.

Moreover, requirements engineering plays an important role during the later stages of the development process. This is due to the fact that the requirements of the system might determine the technology that is to be used for the system design and implementation. This approach, widely known as requirements driven development approach, aims to analyse the requirements of the system-to-be and determine, based on this analysis, if a technology is suitable for the development of a particular system.

[1] The term stakeholders refer to anyone who might be affected by the system and who have an influence on the system requirements. This might include organisations, users, managers, customers, and authorities.

2.1.2 Design stage

When the requirements analysis phase is completed and an accurate description of the requirements of the system has been produced, the next stage involves the transformation of those requirements to design.

The belief of this research is that a design essentially forms a complete description of the system-to-be, which must be independent of implementation platforms. As a result, the design stage is considered as important as the requirements analysis phase since a well-designed system is easy to understand, implement and maintain.

Design involves the specification of the system's software architecture and the components within the system. Therefore, the design stage must define explicitly the architecture of the system as a whole as well as the individual components of it. In addition, the complexity of the design, even if the system is quite complex, must be kept manageable. To do this, the design stage must provide techniques to decompose the complexity of the system and thus make the final design easier to understand.

Although this research treats the requirements analysis and design stages as two separated phases of the software engineering process, it also considers them closely related. This is due to the fact that the requirements analysis stage should be in consistency with the design stage, and one should fulfil the other. This is very important since producing a design that is in inconsistent with the requirements means the developed system will not operate according to the user needs.

2.1.3 Development methodologies

In both the requirements analysis and the design stages developers use guidelines, notations and follow structured processes to help them go through these stages faster and more efficiently. In other words, they are using methodologies (guidelines, structure processes) and modelling languages (notations) to analyse and design a software system.

To emphasise the need of employing a methodology in the development of a system, Birrell and Ould argue [Bir86] "*anyone undertaking software development, on no matter what scale, must be strongly advised to establish a methodology for that development –one or more techniques that, by careful integration and control, will bring order and direction to the production process*".

According to Booch [Boo94] *"a methodology is a collection of methods applied across the software development life cycle and unified by some general, philosophical approach"*.

Hubmann [Hub97] argues that a methodology always consists of the following four components:

- A definition of the problem space to which the methodology is applicable.
- A set of models that represent different aspects of the problem domain or the solution at different stages.
- A set of methods that transform instances of one model into another model.
- A set of procedural guidelines that define an order for the systematic application of the methodological steps.

On the other hand, Russel claims [Rus00] that a methodology usually consists of two parts, a modelling language (that forms the ontology of the methodology), which is usually graphical, and a collection of integrated techniques that help in the analysis and the design.

A modelling language is effectively a collection of elements that helps to model and document the system. As a result, a modelling language gives the designer the opportunity to develop a system without limiting the creativity with any constraints of a particular programming language. Furthermore, a graphical representation of the system presents a much clearer idea of the system than a programming language. A well-known modelling language is the Unified Modeling Language (UML) [Fow00].

On the other hand, an integrated technique provides a set of well-defined steps that gives developers the opportunity to split the system in several sub-systems making the analysis and design easier. It must be noticed that there are different techniques for the analysis and the design phases. Analysis techniques help to develop models of *why* and *what* is required from the system, whereas design techniques help to model *how* the system will achieve its requirements.

Although Russel's argument is true for most of the analysis and design methodologies especially the well-known ones such as the Object Modelling Technique (OMT) [Rum91] and Booch [Boo94], the author of this thesis believes that it cannot be taken as a general rule for all the methodologies.

In this research, ***a development methodology is considered as a pre-defined series of steps that helps developers to understand a problem and model a solution***. As a result, an analysis and design methodology should provide methods, guidelines, descriptions, and tools for each of the analysis and design phases in the life cycle of a system. In addition, a good analysis and design methodology should identify errors and encourage modifications at the earliest possible time of the analysis and design phases. Although a methodology must guide through its steps, at the same time it must be flexible and allow creativity. In other words, although a methodology must be well defined, it should not dictate every aspect of the development.

2.2 AGENT ORIENTED SOFTWARE ENGINEERING

2.2.1 Agents and multiagent systems

Agent oriented software engineering is based on the concept of an agent[2]. The term agent derives from the present particle of the Latin verb *agere*, which means to drive, act, lead or do [Bra97]. Although the term *software agent* is widely used, there is not a standard definition of what is a software agent. According to Nwana [Nwa96], "*there are at least two reasons why it is so difficult to define precisely what a software agent is.*"

A first reason is the fact that the word *agent* is not owned by the software agent researchers. It is a term that it is widely used outside the agent community. According to the Cambridge International Dictionary of English the word agent means "*a person who acts for or represents another*". Thus, the term is widely used in estate agents, or travel agents just to name a few of the cases.

A second reason is that a software agent can play many roles. There are software agents that help to navigate, to search, or even software agents that can act as personal assistants. Therefore, inside the agent community the term *agent* has different definitions for different people. Wooldridge & Jennings define an agent as [Wool95] "*.... A hardware or (more usually) software based computer system that enjoys the following properties*:

[2] In this thesis the term agent refers always to a software agent.

1. ***Autonomy***. Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

2. ***Social ability***. Agents interact with other agents (and possibly humans) via some kind of agent communication language.

3. ***Reactivity***. Agents perceive their environment, (which may be the physical world, such as a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.

4. ***Pro-activeness***. Agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour by taking the initiative".

On the other hand, according to IBM (as quoted in [Fra96]) " *intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires*", whereas according to P. Maes [Mae95] agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

Throughout this thesis, the definition of Wooldridge and Jennings is used, that is, an agent represents a software having properties such as autonomy, social ability, reactivity, and proactivity.

Despite the many different definitions, it is widely agreed that the true power of the agent paradigm is realised from the use of multiagent systems[3]. These are systems that contain more than one software agent. In a multiagent system a task is divided into a set of subtasks and distributed amongst the different software agents of the system.

The area of multiagent systems has its roots in several disciplines with the two most important and relevant being the "Distributed Artificial Intelligence (DAI)" and

[3] In fact it has been stated that "*it can been argued that there is no such thing as a single agent system; everything involves multiple agents*" [Jen99].

"Artificial Life (AL)". The former deals with creating systems capable of solving problems and the latter tries to understand and model systems possessing life.

The influence of "Distributed Artificial Intelligence" field to the multiagent systems came mainly from one of the most important figures in the Artificial Intelligent (AI) field, C. Hewitt. Hewitt [Hew77] used in his research active entities called actors. He thought of the idea of problem solving as an activity of many different expert individuals. This thought gave birth to ideas such as languages for actor communications. These languages are still considered as being good bases for the creation of multiagent systems [Fer99].

The area of "Artificial Life" has influenced the multiagent systems research since it provided the underlying principles of the organisation of living things. These principles are now being studied and tested in a computer environment, giving very useful results for the multiagent systems research community.

Later than Hewitt, Kinny et al. [Kin96] claimed that in specifying a multiagent system, it is highly desirable to adopt a more specialised set of models, which operate at two distinct levels of abstraction. In the first level (external viewpoint) the system is decomposed into agents modelled as complex objects characterised by their purpose, their responsibilities, the services they perform, the information they require and maintain, and their external interactions. In the second level (internal viewpoint) the individual elements required by the particular agent architecture, such as beliefs and goals, must be modelled for each software agent of the multiagent system.

2.2.2 *Defining the agent oriented software engineering paradigm*

Work within the agent research community has lead towards the development of agent oriented software engineering (AOSE) paradigm. AOSE introduces an alternative approach in analysing and designing complex distributed computerised systems [Jen01, Wool01, Igl99], according to which a complex computerised system is viewed as a multiagent system [Wool01] in which a collection of autonomous software agents (subsystems) interact with each other in order to satisfy their design objectives. Therefore, developers view the system as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, intelligence and the capability of communicating [Mou03].

2.2.2.1 Agents as a modelling construct

Most of the current work in agent oriented software engineering originated from the programming and the AI/DAI systems constructions perspective [Yu01]. As a result, early work in Agent oriented software engineering was focused around the concept of an agent as a concrete artefact rather than a modelling construct.

However, an important point of the agent-oriented software engineering is that its use for the analysis and design of a system does not necessarily impose the use of agents as the implementation choice. Towards this direction, efforts have been made, in the last few years, to define the concept of an agent as a modelling construct rather than a concrete artefact. Yu [Yu01a] proposes that the concept of an agent as a modelling construct should have the following properties: autonomy, intentionality, sociality, identity and boundaries, strategic reflectivity and rational self-interest. Although most of these properties have been mentioned earlier in the various definitions of the term agent (section 2.2.1), their significance when considering agents as a modelling paradigm is quite different [Yu01].

Autonomy does not refer to the ability of a software agent to act without the direct intervention from humans, but rather to the adoption (from the view point of developers) of a less simplistic view of the world, in which uncertainties are taken into account when considering possible different alternatives for achieving a system's objectives.

Intentionality allows developers to provide a higher-level description of the behaviour of the components of a system by employing intentional concepts such as goals, tasks, beliefs and capabilities.

Sociality refers (from the developers' point of view) to the modelling of the different agents of a system in terms of their relationships, commitments and dependencies. The property of sociality allows the better definition of these relationships since it allows the creation and usage of new and close to real world abstractions, such as actors, roles, and positions, to guide the development of a system.

The property of identity refers to the perception of a software agent. A software agent as a modelling entity is not necessarily a physical agent but rather an abstract entity that exhibits agent behaviour. It is up to the developer to distinguish between

the physical and the abstract entities that will constitute the system. In addition, the boundaries of an agent are contingent and changeable according to the relationships, dependencies and commitments that the agent participates.

Strategic reflectivity refers to the process of reasoning about design choices by considering different alternative ways rather than modelling a specific way. This process is strategic because agents (abstract agents and not physical) determine which alternatives would better serve their strategic interests.

Rational self-interest means developers try to model the preferences and the decisions of the system's stakeholders in terms of those options that best serve their interests. This allows drawing conclusions (limited sometimes) about their behaviour in the system.

2.2.2.2 The arguments for the use of agent oriented software engineering paradigm

It is early to say that the agent oriented software engineering (AOSE) paradigm will become widely successful, since no evidence yet exists to suggest that agent-oriented software engineering will actually improve the software engineering productivity. However, there are convincing arguments for believing that agent orientation will be of benefit for engineering certain complex software systems [Jen99].

According to Jennings and Wooldridge [Jen99], there are three main arguments for an agent oriented approach. First of all, the effectiveness of agent oriented decompositions in partitioning the problem space of a complex system. A complex computerised system can be decomposed into smaller components, the same way that a multiagent system can be decomposed into the elements that constitute the system (software agents). Secondly, the suitability of the key agent oriented abstractions, such as agents, (social) interactions and organisations, in modelling complex systems, and thirdly the appropriateness of the agent oriented philosophy for dealing with the dependencies and the interactions that exist in a complex system.

Furthermore, as Lind [Lin01] notes, agent oriented software engineering provides "*an epistemological framework for effective communication and reasoning about complex software systems on the basis of mental qualities. It provides a consistent*

new set of terms and relations that adequately capture complex systems and that support easier and more natural development of these systems".

To the above points argued by Wooldridge, Jennings and Lind, this thesis adds that the factor that really makes agent oriented software engineering distinct from any other software engineering paradigm is the higher level of abstraction employed in the development of software systems. The idea of modelling a system in terms of autonomous entities with characteristics similar to humans introduces a close-to-real-life modelling of the system, and therefore makes the development of the software system natural.

The higher level of abstraction that agent oriented software engineering introduces, together with the reasoning in terms of mental qualities that Lind discusses, provides a software engineering paradigm that naturally helps to narrow the gap between real life and modelling, by allowing developers to reason about the software system using concepts and mental qualities known to them from the real life.

However, as mentioned by Kinny et al. [Kin96], *"if agent oriented software engineering is to become widely accepted as a paradigm for the development of large-scale applications, adequate agent-oriented methodologies and modelling techniques will be essential. This is not just to ensure that systems are reliable, maintainable, and conformant, but to allow their design, implementation, and maintenance to be carried out by software analysts and engineers rather than researchers".*

This argument summarises what is well known within the agent research community [www.agentlink.org]: The existence of mature and complete methodologies, to help developers to model software systems by taking into account the unique characteristics that agent orientation introduces, is an important issue for the success and wide acceptance of agent oriented software engineering.

The next section provides a discussion with respect to agent oriented software engineering methodologies.

2.2.3 Agent oriented software engineering methodologies

Attempts to develop agent oriented software engineering methodologies have been mainly divided into three categories. Those inspired by object oriented

methodologies, those that consider knowledge engineering methodologies and those that have been developed with agent orientation in mind.

This diversity has naturally raised the question *if the current methodologies, which are customised to object oriented systems* (see for instance [Boo94, Fic98, Eri98, Jac99]), *knowledge-based systems* (see for instance [Schr99]) *or another, can be used as agent oriented software engineering methodologies, if they need to be extended or slightly change or if they are totally inappropriate to help the analysis and design of systems with agent orientation in mind.*

An answer to such a question is not simple. On one hand, current methodologies are based on software engineering rules and ideas, and as mentioned in section 2.1.3, any methodology must follow some basic rules and ideas of software engineering, independent of the paradigm used. Therefore, considering the question only from this point of view, the answer is that the object oriented or knowledge engineering methodologies can indeed be used for the development of agent oriented systems.

On the other hand, the different ideas and characteristics of each paradigm must be taken into consideration. For instance, the different level of abstraction employed by the agent oriented software engineering in analysing and designing complex systems introduces characteristics that object oriented and knowledge engineering based methodologies fail to adequately model. Because of this, it has been argued [Wool00], that "*if agents are to realise their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are specifically tailored to them*". From this point of view, the answer to the above question is that non agent oriented methodologies are inappropriate when considering an agent oriented view of a system.

Nevertheless, both of the views just described are quite extreme, either black or white. This is not always the case and in providing a mature answer to the given question, the rest of this section provides a discussion on the suitability of object oriented and knowledge engineering methodologies for the development of complex systems with agent orientation in mind.

As mentioned by Iglesias [Igl99], several reasons can be cited that justify the extension of current object oriented methodologies for the development of systems with agent orientation in mind. These include, the similarities that can be found

between the two main concepts, namely object and agent [Wool01, Kin96], the commonly usage of object oriented languages, such as JAVA, for the implementation of agent oriented systems [Igl99], and the familiarity of many software engineers with object oriented methodologies [Igl99]. In addition, both the object oriented and the agent oriented paradigms emphasize the importance of interactions between the entities of the system [Jen99].

However, many shortcomings can be identified on the extension of current object oriented methodologies for the development of software systems with agent orientation in mind.

When employing agent orientation in the development of a system, the system is modelled consisting of entities (agents) that are autonomous, and have intentions. Therefore, techniques and models are required to model these characteristics. However, the level of abstraction and the models provided by object oriented software engineering methodologies are not adequate to model these characteristics.

Another crucial difference is that, in the object oriented paradigm, there is no programming construct that supports the realisation of a subsystem, whereas in the agent oriented paradigm, software agents are used to realise particular instances of roles, which then take on a separate identity and existence [Zam01].

Furthermore, object oriented software engineering fails to provide an adequate set of concepts and mechanisms for modelling complex systems [Jen01]. As mentioned by Booch [Boo94] "*for complex systems we find that objects, classes and modules provide an essential yet insufficient means of abstraction*". The Object Model that is the primary specification [Wool00] of an object oriented system, fails to capture the dynamic nature of the interactions between the agents since it captures static dependencies and paths of accessibility which are irrelevant in multiagent systems [Wool00].

In addition, according to Wooldridge and Ciancarini [Wool01] "*object oriented methodologies consist of an iterative refinement cycle of identifying classes, specify their semantics and relationships, and elaborating their interfaces and implementation. At this level of abstraction, they appear similar to typical agent oriented methodologies, which usually proceed by identifying roles and their responsibilities and goals, developing an organizational structure, and elaborating*

the knowledge and behaviours associated with a role or agent. However this similarity disappears at the level of detail required by models, as the key abstractions involved are quite different. For example, the first step of object class identification typically considers roles, organizations, events and even interactions as candidate objects, whereas these need to be clearly distinguished and treated differently in an agent-oriented approach". The point made by Wooldridge and Ciancarini is very important since such a distinction is essential in order to model the sociality property (section 2.2.2.1) of the agents of the system.

Moreover, object oriented methodologies lack of models and techniques to capture the metal states (such as goals, tasks and capabilities) of the agents of a system, as well as the social relationships that the agents demonstrate in a multiagent system environment.

Apart from object orientation, another paradigm that agent researchers use as the basis for developing analysis and design methodologies for agent oriented systems is knowledge Engineering (KE).

KE methodologies (see for instance [Schr99]) are used for the analysis and design of knowledge-based systems. The main argument for the usage of knowledge engineering methodologies is that most of the problems, such as knowledge acquisition, modelling and reuse, subject to knowledge engineering methodologies are present in the development of systems with agent orientation in mind. Therefore, knowledge engineering methodologies can provide the techniques for modelling the knowledge of the agents of the system. In addition, both the existing tools and the developed ontology libraries and problem solving method libraries can be reused [Igl99].

On the other hand, the main limitation comes from the fact that although these methodologies can provide techniques for modelling in detail the knowledge of the different agents that are included in a multiagent system, they fail to capture the autonomous, intentional and social behaviour of these agents. In addition, most of the knowledge engineering methodologies lack flexibility and as a result, it is difficult to adequately extend them to capture agent concepts.

A third category of agent oriented software engineering methodologies includes methodologies specifically developed with agent orientation in mind. Efforts towards

this direction have grown rapidly the last few years, and as a result many methodologies based on the agent oriented software engineering paradigm have been developed (see [Igl99] for a review and [Eva01, Giu02, Igl97, Wood01, Wool99] for more details of some of the methodologies). The main advantage of these methodologies is the inclusion of models and notations to capture all the unique characteristics that agent orientation introduces. In particular, such methodologies can model the system in terms of agents that have properties such as autonomy, intentionality, identity and boundaries, strategic reflectivity and rational self-interest, and as a result take full advantage of the abstraction that agent oriented software engineering provides when developing systems with agent orientation in mind. However, one of the main disadvantages is that, as with any new methodology, time is required before the methodology can be considered mature and complete.

From the above discussion this research concludes that object oriented or knowledge engineering methodologies are inappropriate, as they are, to adequately model software systems with agent orientation in mind. The best solution is to develop new methodologies tailored to agent oriented software engineering, but at the same time, the knowledge obtained from the object oriented, knowledge based or other methodologies should be taken into consideration. This means, that agent oriented software engineering methodologies should be able to adopt, where suitable, existing methods and take advantage of the work that has taken place in the fields of object oriented and knowledge engineering methodologies.

2.3 SECURITY MODELLING

2.3.1 Basic security concepts and ideas

Physical security systems have been around for many thousands of years, ranging from castle fencing, to window bars and door locks. Computer security, on the other hand, although newer in comparison with physical security is definitely not a new topic since its history starts in the 1960s [Sal75]. Nevertheless, it was until the advent of distributed systems and computer networks that security of software systems has become an issue of huge concern.

As software systems, agent oriented, object oriented or otherwise, become more and more critical in every aspect of human society, from the health sector to military,

so does the demand to secure these systems. This is because private information is stored in computer systems and without security, organisations are not willing to share information or even use the technology.

Take as an example a health and social care information system containing health data of different individuals. Security in such a system, as in any health and social care information system, is very important since security breaches might result in medical history to be revealed and this could have serious consequences for particular individuals.

Security of computer based information systems is concerned with methods providing cost effective and operationally effective protection of information systems from undesirable events [Lan85]. Thus, security is usually defined in terms of the existence of any of the following properties:

- **Confidentiality**: The property of guaranteeing information is only accessible to authorised entities and inaccessible to others.
- **Authentication**: The property of proving the identity of an entity.
- **Integrity**: The property of assuring that the information remains unmodified from source entity to destination entity.
- **Access Control**: The property of identifying the access rights an entity has over system resources.
- **Non repudiation**: The property of confirming the involvement of an entity in certain communication.
- **Availability**: The property of guaranteeing the accessibility and usability of information and resources to authorised entities.

Failure of any of the above-mentioned security properties might lead to many dangers ranging from financial losses to sensitive personal information losses. The existence of the above security properties within a system is defined in terms of the security policy. A security policy can be defined as "*the set of rules that state which actions are permitted and which actions are prohibited*" [Gol01]. A security policy determines the limits of acceptable behaviour and what the response to violations should be and it might define possible mechanisms, widely known as security mechanisms, designed to detect, prevent or recover from a security attack. A security

attack is defined [Sta99] as an action that compromises the security information owned by an organisation.

According to Anderson [And01], "*security engineering is about building systems to remain dependable in the face of malice, error or mischance*". To design a secure system it is important to know what the potential threats are so that appropriate counter-measures can be taken. However, no matter how good the protection is, possible attackers will (and have up to now) find possible vulnerabilities to expose the system. In addition, during the analysis and design the developer assumes the infrastructure is 100% trustworthy. However this might not be the case, making the prediction of every possible attack during the development of the system impossible, and allowing a potential attacker to attack the system with types of attack that the developer cannot identify during the development of the system.

Because of this, a well-known axiom of computer security states that the only completely secure computer system is the one that has never been turned on. Therefore, usually the goal will be to provide as much security as possible trading sometimes security requirements with other functional and non-functional requirements.

2.3.2 Security in software engineering

A security requirement is defined as "*a manifestation of a high-level organisational policy into the detailed requirements of a specific system*" [Dev00]. Agent oriented software engineering considers security requirements as non-functional requirements [Chu95]. Non-functional requirements introduce quality characteristics, but they also represent constraints under which the system must operate [Rom85, Som01]. Although software developers have recognised the need to integrate most of the non-functional requirements, such as reliability and performance, into the software development processes [Dar91]; security still remains an afterthought.

Therefore, the usual approach towards the inclusion of security within a system is to identify security requirements after the definition of a system or consider security only in certain stages of the development process. However, these approaches often lead to problems [And01], since security mechanisms have to be fitted into a pre-existing design, therefore leading to serious design challenges that usually translate

into software vulnerabilities [Sta99]. Literature provides many examples of security disasters that happened while trying to upgrade a non-secure system to a secure system (see for instance [Bay95]).

Thus, this research argues that security should be considered during the whole development process and it should be defined together with the requirements specification. By considering security only in certain stages of the development process, more likely, security needs will conflict with functional requirements of the system. Taking security into account along with the functional requirements throughout the development stages helps to limit the cases of conflict, by identifying them very early in the system development, and find ways to overcome them. On the other hand, adding security as an afterthought not only increases the chances of such a conflict to exist, but it requires huge amount of money and valuable time to overcome it, once they have been identified (usually a major rebuild of the system is needed). This argument has also been supported many times in the literature [Dev00, Jur01, Try97].

However, to consider security issues throughout the development process of a software system, software engineering methodologies must provide developers with models and processes to help them model security concerns. Nevertheless, as mentioned in section 1.4, current methodologies do not meet the needs for resolving the security related problems [Try97], and fail to provide evidence of integrating successfully security concerns throughout the whole range of the development process. In other words, they fail to adequately provide a security-oriented approach in the development of software systems.

2.3.2.1 Problems of modelling security during the development of a system

The development and the definition of such an approach is a demanding and difficult task. It is demanding because there are many problems associated with the consideration of security issues during the analysis and design stages that must be overcome and difficult because there are requirements that such a security-oriented approach must satisfy. The aim of this section is to discuss the problems associated with the consideration of security issues during the whole development process.

Integrating security issues within the development stages of a development methodology is difficult mainly due to the following reasons, [Mou03a, Mcd99, Chu95, Schu01]:

1. Developers, who are not security specialists, usually need to develop multiagent systems that require knowledge of security;
2. Many different concepts are used between security specialists and software engineers. As a result, there is an abstraction gap that makes the integration of security and software engineering more difficult;
3. There is an ad hoc approach towards security;
4. It is difficult to define together security and functional components and at the same time provide a clear distinction. For instance, which components are part of the security architecture, and which ones are part of the functional specification;
5. It is difficult to move from a set of security requirements to a design that satisfies these requirements, and also understand what are the consequences of adopting specific design solutions for such requirements;
6. It is difficult to get empirical evidence of security issues during the design stage. This makes the process of analysing security during the design stage more difficult;
7. It is difficult to fully test the proposed solutions at the design level;

2.3.2.2 Requirements of a security-oriented approach

To successfully overcome the above-mentioned problems, a security-oriented approach should comply with the following requirements:

1. Must allow novice security developers to successfully consider security issues during the analysis and the design of multiagent systems (response to problem 1).
2. Must employ the same concepts and notations during the whole development process (response to problem 2).
3. Must be integrated within a methodology. The guidelines and the structural processes of the methodology will allow the explicit definition of the

applicability of the security process within the stages of the methodology (response to problem 3).

4. Must be clear and well guided (response to problem 3).
5. Must provide means to check that the development process is consistent (response to problem 3).
6. Must define together security and functional requirements but also provide a clear distinction (response to problem 4).
7. Must allow developers to identify possible conflicts between security and other functional and non-functional requirements (response to problem 4).
8. Must allow developers to understand the consequences of the application of a particular design (response to problem 5).
9. Must allow developers to move to a design that successfully satisfies the security requirements (response to problem 5).
10. Must allow developers to analyse security requirements and base design solutions on such an analysis. In other words, it should allow developers to explore different architectural designs according to the identified security requirements (response to problem 6).
11. Must allow developers to evaluate the developed security solution (response to problem 7).

2.4 AGENT ORIENTED SOFTWARE ENGINEERING AND SECURITY ENGINEERING

The agent oriented software engineering paradigm presents a feasible approach for the integration of security to software engineering. This is mainly due to the appropriateness of agent oriented philosophy, for dealing with the security issues that exist in a computer system.

Security requirements are mainly obtained by analysing the attitude of the organisation towards security and after studying the security policy of the organisation. As mentioned in [Jen99] agents act on behalf of individuals or companies interacting according to an underlying organisation context. The integration of security within this context will require for the rest of the subsystems (agents) to consider the security requirements, when specifying their objectives and

interactions therefore causing the propagation of security requirements to the rest of the subsystem.

In addition, the agent oriented view is perhaps the most natural way of characterising security issues in software systems. Characteristics, such as autonomy, intentionality and sociality, provided by the use of agent orientation allow developers first to model the security requirements in high-level, and then incrementally transform these requirements to security mechanisms.

However, as mentioned in chapter 1, none of the existing agent oriented software engineering methodologies have demonstrated enough evidence to support claims of adequately integrating security during the whole development process.

2.4.1.1 Identification of a suitable methodology

As mentioned in the Introduction, this research project aims to extend an agent oriented software engineering methodology, rather than developing one from scratch, to enable it to model security issues throughout the development lifecycle.

As a result, different methodologies were compared in order to identify the one that is most suitable for this project. During this evaluation/comparison the following criteria[4] were used.

1. **Support.** Is the methodology well supported? Is material related to the methodology published? Are there any tools available?
2. **Accessibility.** Are the models and the processes of the methodology easily understandable?
3. **Expertise.** Does the methodology assume knowledge/expertise in a particular discipline?
4. **Implementation-targeted.** Is the methodology restricted to a particular implementation choice?
5. **Development coverage.** How much of the development lifecycle the methodology covers?
6. **Extensibility.** Is the methodology easily extensible?

[4] Some of these criteria are loosely based on criteria proposed by Sturm and Shenory [Stu03].

7. Security-aware. Does the methodology consider any security issues within its development processes and models?

To evaluate the methodologies a scale of 1-4 has been decided, where 1 indicates the methodology does not address the specific property, 2 indicates the methodology partially addresses the specific property, 3 indicates that the methodology addresses the specific property but some minor deficiencies still exist, and 4 indicates the methodology fully addresses the specific property.

From a large amount of existing agent oriented software engineering methodologies (see [Igl99] for a review and [Eva01, Giu02, Igl97, Wood01, Wool99] for more details on some of them), four methodologies were chosen and compared, namely GAIA, Tropos, MaSE and MAS-CommonKADS.

The following paragraphs provide the reasons for choosing these methodologies, and a brief introduction to each of them together with references for readers interested in obtaining more information about these methodologies. It must be noticed that the aim of these paragraphs is not to provide a detail description of these methodologies; this is out of the aim of this section.

The GAIA methodology [Wool00, Zam01] was chosen because it is a well-known methodology developed by leading researchers in the field of software agents. The methodology deals with both the societal (macro) level and the agent (micro) level aspects of the design [Wool00] and it borrows some terminology and notation from the FUSION [Col94] object oriented methodology. Nevertheless, it is not just an agent based extension of the FUSION. GAIA was developed having in mind that most of today's analysis and design methodologies fail to capture the complexity of an agent system's organisational structures as well as the flexibility of agents. It is worth mentioning that the methodology views the requirements phase as separate from the analysis and design phases. In the analysis phase the system is identified using the notion of organisation, whereas design aims to transform the analysis models into a sufficiently low level of abstraction that traditional design techniques may apply in order to implement the agents.

Tropos [Giu02, Bre02, Bre02b] was chosen because it is a widely known and published agent oriented software engineering methodology and one of the few that provides some kind of security modelling. Tropos is a requirements driven

methodology that describes both the environment of the system and the system itself. Its main advantage is that it covers the whole development process, from the early requirements to design, using the same concepts and notations. Tropos adopts the i* modelling framework [Yu95], which uses the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations of actors to other actors.

The Multi-agent Systems Engineering Methodology (MaSE) [Sco01, Sco02, Wood01] was chosen, although it is similar to the GAIA methodology, because it is more specialised than GAIA for its use in the distributed agent paradigm and goes further by providing support for generating code using the MaSE code generation tool (http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm). One of the main differences between this methodology and other agent based methodologies is that in the MaSE methodology the general components of the system are designed before the system itself is actually defined. Although the diagrams of the methodology might look similar to Unified Modelling Language (UML) diagrams, they have been modified in order to model notions of agents as well as their cooperative behaviour.

MAS-CommonKADS (MAS-CK) [Igl97] was chosen because forms an extension of the knowledge engineering CommonKADS methodology [Schr99] that is considered a European standard for knowledge modelling. This methodology extends the CommonKADS methodology by adding object oriented techniques. It also "borrows" protocol engineering in order to define the agent protocols. Apart from the analysis and design phases, the methodology also provides a conceptualisation phase, in which the user requirements and a first description of the system are defined. The conceptualisation phase is the first step in the MAS-CommonKADS methodology. Then the methodology defines models for analysing and designing a system. In each of these models the methodology defines the "constituents" (entities to be modelled) and the relationships between these entities. It must be noticed that the process is "risk driven". That is, "*in every cycle the states of the models to be reached are defined by reducing the perceived risks*" [Igl97].

Table 2-1 indicates the evaluation of the above methodologies with respect to the evaluation criteria defined in the beginning of the section.

Support. Although the GAIA methodology is well known, it is not well supported. There are only two main papers [Wool99, Wool00] that describe the methodology and there are no automatic tools or any support group. On the other hand, the Tropos methodology is an international project and support is provided either through the Tropos project [http://www.troposproject.org] or through the many papers published about Tropos. However, tool support is provided only in the form of a diagram editor [Bre03]. Although, there is no support group for the MAS-CommonKADS, information about the methodology can be found in terms of the publications [Igl96, Igl97, Igl97a] related to the methodology. There is no tool support for the methodology, although the developers claim that they are working towards the development of a tool [Igl96]. The MaSE methodology is supported by the group members of the Multiagent and Cooperative Robotics Lab (see the web page in http://www.cis.ksu.edu/~sdeloach/ai/mase.htm) and information can be found in terms of many publications about the methodology [Sco01, Sco02, Sel03, Wood01]. In addition, a tool, as mentioned above, exists to support the methodology.

Table 2-1:Evaluation of the methodologies

Property/ methodology	GAIA	TROPOS	MAS-CK	MaSE
Support	1	3	1	4
Accessibility	3	3	3	3
Expertise	2	2	2	2
Implementation-targeted	4	4	4	4
Development Coverage	2	4	3	3
Extensibility	4	4	3	4
Security Aware	1	2	1	1

Accessibility. All of the above mentioned methodologies have models and processes that are well defined and relatively easy to understand. However, some background knowledge is required by all the methodologies.

Expertise. Although GAIA is relatively easy to understand, it requires some knowledge in logic and temporal logic. This can be a substantial drawback for people without the appropriate knowledge. Tropos, on the other hand, requires some background in requirements engineering and analysis techniques such as goal

analysis, decomposition, and means-ends analysis. MAS-CommonKADS requires background knowledge related to knowledge engineering as well as use-cases engineering, whereas MaSE mainly requires knowledge of object oriented analysis and design techniques, such as OMT, modelling languages, such as UML, and use-cases engineering.

Implementation-targeted. None of the presented methodologies are targeted towards a particular implementation choice.

Development coverage. With the exception of the Tropos methodology, which covers the whole development process, from the early requirements analysis to implementation, the other three methodologies cover only specific parts of the development process. GAIA only considers analysis and design, ignoring requirements and implementation stages. On the other hand, both MAS-CommonKADS and MaSE start their development processes from the late requirements analysis missing the early requirements analysis stage.

Extensibility. GAIA, Tropos and MaSE allow improvements and extensions to be made with relative ease. On the other hand, the fact that MAS-CommonKADS is based on concepts from knowledge engineering makes it a bit more difficult to apply any extensions regarding agent oriented concepts.

Security Aware. The only methodology that provides some kind of support for security modelling is the Tropos methodology. Tropos employs the concept of soft-goal to model some security issues [Mou02]. However, the security modelling provided is very limited and the methodology fails to provide a security-oriented approach in the development of multiagent systems.

Taking into account the above evaluation, the Tropos methodology was chosen as the methodology to be extended to enable the modelling of security issues throughout the development process. This decision was mainly based on the fact that Tropos spans in all the development stages using the same concepts, it is easily extensible and also it is more security aware than the other methodologies. In addition, the Tropos methodology is well integrated with other approaches, such as the UML, in which some security work has taken place [Jur01, Jur02, Lod02], and therefore existing work can be considered and incorporated within the proposed approach. Moreover, the modelling concepts of Tropos are well suited to model

security requirements, which are usually expressed using notions such as agents and high level goals such as confidentiality and authentication [Gio03].

2.5 SUMMARY

This chapter aimed to establish a common language for the understanding of the next chapters and discuss issues that form the basis for the achievement of the aims of this thesis.

Thus, this chapter defined software engineering and it provided discussions regarding requirements engineering, the design stage of the development process, and software development methodologies. Moreover, agents, multiagent systems and the agent oriented software engineering paradigm were defined, and a critical discussion was presented regarding agent oriented software engineering methodologies. In addition, this chapter discussed security modelling by providing basic security concepts and ideas, and by examining how security is considered within agent oriented software engineering. It then argued the necessity to consider security issues during the whole development lifecycle.

Moreover, an outline of the problems of modelling security during the development of a system was given and a minimum set of requirements that a security-oriented approach should meet was proposed.

This chapter also identified the Tropos methodology as the appropriate methodology to proceed in this research project. Thus, it is important before describing the proposed security extensions to provide a detailed description of the Tropos methodology. The next chapter provides such description.

CHAPTER 3 AN OVERVIEW OF THE TROPOS METHODOLOGY

Chapter 2 provided a discussion of (some) development methodologies for multiagent systems and it identified the Tropos methodology as the candidate methodology for integrating a security-oriented approach during the development of multiagent systems. However, before describing how the Tropos methodology can be extended to enable the development of multiagent systems with security in mind, it is necessary to provide an overview of the Tropos methodology and also provide a critical discussion of its limitations with respect to security modelling. These are the aims of this chapter.

Section 3.1 provides a basic introduction to the Tropos methodology indicating its advantages and its key features. Section 3.2 reviews the main concepts and notations of the Tropos methodology and section 3.3 defines the Tropos development stages. Moreover, the modelling language of the methodology is described in section 3.4 and the Tropos modelling activities are introduced in section 3.5. In addition, a set of transformations, which enable developers to refine the development models, is described in section 3.6 and section 3.7 presents the methodology with the aid of an example. Section 3.8 discusses the limitations of the Tropos methodology when modelling security issues during the development of multiagent systems and section 3.9 summarises the chapter.

3.1 AN INTRODUCTION TO THE TROPOS METHODOLOGY

Tropos[5] is a novel agent oriented software engineering methodology tailored to describe both the organisational environment of a multiagent system and the system itself. Tropos is a requirements driven methodology, in the sense that it is based on concepts used during early requirements analysis, such as actors, goals and tasks, and

[5] The name Tropos derives from the Greek "Τρόπος" which means "way of doing things" but also has the connotation of "easily changeable, easily adaptable".

the novelty of the methodology lays on the fact that those concepts are used to model not just early requirements, but also late requirements as well as architectural and detailed design [Cas02]. Using the same concepts during the development stages of a multiagent system provides the advantage of reducing impedance mismatches between different development stages, and therefore streamlines the development process [Cas02].

Tropos is characterised by three key aspects [Cas02, Per01, Giu02, Bre02b]. Firstly, it deals with all the phases (requirements analysis, system design and implementation) of a system development, adopting a uniform and homogeneous way that is based on the notion of agents and all the related mentalistic notions, such as actors, goals, tasks, resources, and intentional dependencies. According to Bresciani et al. [Bre02b], the decision to use mentalistic notions in all the phases of analysis has important consequences, since it helps to reduce to a minimum the conceptual gap from *what* the system must do and *why*, and *what* the users interacting with it must do and *why*. This provides (part of) the flexibility needed to cope with multiagent application's complexity. Secondly, Tropos pays a great deal of attention to the early requirements, emphasizing the need to understand not only *what* organisational goals are required, but also *how* and *why* the intended system would meet the organisational goals. This allows for a more refined analysis of the system dependencies, leading to a better treatment not only of the system's functional requirements but also of its non-functional requirements, such as security, reliability, and performance [Per01]. Thirdly, Tropos is based on the idea of building a model of the system that is incrementally refined and extended from a conceptual level to executable artefacts, by means of a sequence of transformational steps [Bre02, Bre02a]. Such transformations allow developers to perform precise inspections of the development process by detailing the higher level notions introduced in the previous stages of the development. In addition, since the methodology employs the same notation throughout the development process, such a refinement process is performed in a more uniform way as compared, for example, to UML-based methodologies where the graphical notation changes from one development step to another (for example, from use cases to class diagrams).

It must be noted that Tropos is not a "laboratory" methodology but it has been motivated and illustrated with a number of case studies [Cas02, Bre02b, Mou02a].

3.2 A REVIEW OF TROPOS CONCEPTS AND NOTATIONS

Tropos adopts the *i** modelling framework [Yu95], which uses the concepts of actors, goals and social dependencies for defining the obligations of actors (dependees) to other actors (dependers). This means the multiagent system and its environment are viewed as a set of actors, who depend on other actors to help them fulfil their goals.

An actor [Yu95] represents an entity that has intentionality and strategic goals within the multiagent system or within its organisational setting. An actor can be a (social) agent, a position, or a role. Agents can be physical agents, such as a person, or software agents. In Tropos a classical definition of software agent [Bra97] is used, that is, a software having properties such as autonomy, social ability, reactivity, and proactivity. A role represents an abstract characterisation of the behaviour of a social actor within some specialised context or domain of endeavour [Yu95]. A position represents a set of roles, typically played by one agent. In Tropos, an agent can occupy a position whereas a position is said to cover a role [Bre02b].

A (hard) goal [Yu95] represents a condition in the world that an actor would like to achieve. In other words, goals represent actor's strategic interests. In Tropos, the concept of a hard-goal (simply goal hereafter) is differentiated from the concept of soft-goal. A soft-goal is used to capture non-functional requirements of the system, and unlike a (hard) goal, it does not have clear criteria for deciding whether it is satisfied or not and therefore it is subject to interpretation [Yu95]. For instance, an example of a soft-goal is "the system should be scalable". According to Chung et al. [Chu95], the difference between a goal and a soft-goal is underlined by saying that goals are satisfied whereas soft-goals are satisficed[6].

A task (also called plan) represents, at an abstract level, a way of doing something [Giu02]. The fulfilment of a task can be a means for satisfying a goal, or for

[6] The notion of satisficing assumes that development decisions usually contribute only partially towards (or against) a particular goal, rarely "accomplishing" or "satisfying" goals in a clear-cut sense [Chu95].

contributing towards the satisficing of a soft-goal. In Tropos different (alternative) tasks, that actors might employ to achieve their goals, are modelled. Therefore developers can reason about the different ways that actors can achieve their goals and decide for the best possible way.

A resource [Giu02] presents a physical or informational entity that one of the actors requires. The main concern when dealing with resources is whether the resource is available and who is responsible for its delivery.

A dependency [Yu95] between two actors represents that one actor depends on the other to attain some goal, execute a task, or deliver a resource. The depending actor is called the depender and the actor who is depended upon is called the dependee. The type of the dependency describes the nature of an agreement (called dependum) between dependee and depender. Goal dependencies represent delegation of responsibility for fulfilling a goal. Soft-goal dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely whereas task dependencies are used in situations where the dependee is required to perform a given activity. Resource dependencies require the dependee to provide a resource to the depender. By depending on the dependee for the dependum, the depender is able to achieve goals that it is otherwise unable to achieve on their own, or not as easily or not as well [Yu95]. On the other hand, the depender becomes vulnerable, since if the dependee fails to deliver the dependum, the depender is affected in their aim to achieve their goals.

A capability [Giu02] represents the ability of an actor of defining, choosing and executing a task for the fulfilment of a goal, given certain world conditions and in presence of a specific event.

Figure 3-1 depicts a graphical representation of the above-mentioned concepts as used in the Tropos methodology.

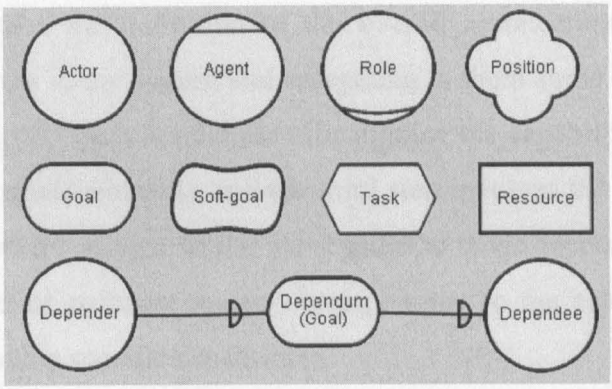


Figure 3-1: Graphical representation of the Tropos concepts

3.3 THE STAGES OF THE TROPOS METHODOLOGY

Tropos methodology covers five main software development stages, starting from the early requirements analysis stage and ending in the implementation stage. Each of these stages is furthered described in the following paragraphs.

During the **early requirements analysis** stage, developers are concerned with the understanding of a problem by studying an existing organisational setting. This involves the identification of the domain stakeholders and their modelling as social actors. In particular, developers model the stakeholders as actors, their intentions as goals, and their relationships as dependencies. Through a goal-oriented analysis [Bre02a], the actors' goals are decomposed into more precise goals and sometimes into tasks that if performed by the actor, allow for goal achievement. The output of this phase is an organisational model, which includes relevant actors and their respective dependencies.

In the **late requirements analysis** stage, the system-to-be is specified within its operational environment, together with relevant functions and qualities. This description models the system as an actor, who has a number of dependencies with the actors identified during the previous stage. These dependencies indicate the obligations of the system towards its environment, and therefore define the system's functional and non-functional requirements.

During the **architectural design** stage, the system's global architecture is defined in terms of subsystems, interconnected through data and dependencies. In particular, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies. This stage is divided into three steps.

The first step includes the definition of the overall architectural organisation by introducing new actors to the system and delegating to them some of the goals of the system. The second step includes the identification of the capabilities needed by the actors to fulfil their goals and tasks and the third step involves the identification of a set of agent types and the assignment of capabilities to those agents. The final output of this stage is a set of software agents corresponding to the actors of the system, each characterised by its specific capabilities.

In the **detailed design** stage, each architectural component is defined in further detail in terms of inputs, outputs, control, and other relevant information. This stage is based on the specifications resulting from the analysis of the previous stages and therefore the reasons for a given element at this stage can be traced back to the early requirements analysis. For this stage, Tropos is using elements of the Agent Unified Modeling Language (AUML) [Bau01] to complement the features of i*.

During the **implementation** stage[7], the actual implementation of the system components takes place according to the design produced in the previous stage. It is worth mentioning that Tropos (as well as other agent-oriented methodologies) does not force the use of Agent Oriented Programming (AOP) as the implementation technology.

3.4 THE MODELLING LANGUAGE OF TROPOS

Tropos defines its own modelling language [Bre02b] in terms of a UML metamodel. The Tropos metamodel is organised into four levels. The meta-metamodel level, which provides the basis for metamodel extensions; the metamodel level, which provides constructs for modelling knowledge level entities and concepts; the domain level, which contains a representation of entities and concepts of a specific application domain; and the instance level, which contains instances of the domain level. For instance, consider an entity as an example of the meta-metamodel, an actor as an example of the metamodel level, a doctor as an example of the domain level and John as an example of the instance level.

[7] This work only considers the first four stages. Implementation is not considered since the proposed security-oriented approach is independent of implementation languages.

The metamodel level of the modelling language allows the more precise (more formal) specification of the Tropos concepts [Giu02]. As an example, consider the concept of actor. Using the Tropos modelling language, an actor is represented as a UML class that can have zero or more (0...*) goals [Bre02b] and zero or more (0...*) beliefs[8] [Bre02b]. Moreover, an actor can depend on another actor (or be the dependee) for a goal, resource, and/or task (plan).

In addition, the meta-metamodel level of the language allows the inclusion of constructs for the formal definition of the Tropos concepts. In particular a formal specification language, called Formal Tropos, is under development [Fux03]. Formal Tropos [Fux03, Fux01] offers all the concepts of graphical Tropos, such as actors, goals and dependencies, supplemented with a rich temporal specification language, inspired by KAOS [Ber98].

3.5 MODELLING ACTIVITIES IN TROPOS

Following the definitions of the levels of the Tropos modelling language, a Tropos model [Bre02b] is defined as a directed labelled graph whose nodes are instances of meta-classes of the metamodel, namely actor, goal, task and resource, and whose arcs are instances of the meta-classes representing relationships (dependencies) between them.

For the development of Tropos models, various activities, such as actor, dependency, goal, task, and capability modelling, and different kinds of graphical diagrams, such as actor, goal, capability and plan diagrams, are used in the Tropos methodology. The rest of this section provides a description of the modelling activities and an introduction to the graphical diagrams of the methodology.

Actor modelling [Giu02] consists of identifying and analysing the system's domain actors as well as the actors of the system together with their goals. During the early requirements stage, actor modelling is focused on identifying the system's domain actors and model them as social actors that have strategic intentions (goals). Later, during the late requirements stage, the system-to-be is introduced as another actor and it is analysed in order to define its functional and non-functional requirements. Actor modelling during the architectural design focuses on providing a

[8] Beliefs represent the actor's knowledge of the world.

more precise definition of the system by decomposing the system into sub-systems (system's internal actors) and on specifying their relationships in terms of data resources and control flows. In detailed design, the actor modelling involves the definition of the system's agents in terms of the notions required by the implementation platform.

As mentioned earlier, sometimes actors depend on each other to accomplish some goals that they would not be able to accomplish (or not in the same degree) without the help of another actor. For this reason Tropos uses **dependency modelling** [Giu02], which involves the identification of the dependencies between the different actors. Dependency modelling spans over the first three Tropos stages namely early and late requirements analysis and architectural design. During the early requirements analysis stage, dependency modelling is focused on identifying dependencies between the actors of the organisation setting in which the system will operate. In late requirements analysis stage, the dependencies between the system and the actors of its organisation setting are identified and some of the actors dependencies identified in the previous stage are refined due to the system introduction. During the architectural design the data and control flows between the different actors of the system are modelled in terms of dependencies providing the basis for mapping the system's actors to software agents.

Goal modelling involves further analysis of particular actors' goals, from the viewpoint of the actor. In other words, the internal goals of each actor identified through actor modelling are furthered analysed in order to provide a more precise definition of the actor. During the early requirements analysis, goal modelling helps to refine the initially identified actors by further analysing their goals and identify new dependencies, or refine existing ones, whereas during the late requirements analysis, goal modelling helps to further analyse the goals of the system. In architectural design, goal modelling motivates the first-decomposition of the system actors into a set of sub-actors [Bre02b].

Soft-goal and **task modelling** are considered complimentary to the goal modelling activity and they employ similar reasoning techniques.

Goal, soft-goal and task modelling are mainly based on three reasoning techniques, *means-end-analysis*, *contribution analysis*, and *AND/OR decomposition*. Means-end

analysis is a mechanism aimed to direct a search process by reducing the difference between a current state and the goal state [Jack90]. In the context of design, means-end analysis drives the design process in a direction that is the shortest distance towards the goal [Sha98]. In Tropos means-end analysis is employed to identify goals, soft-goals, tasks, and/or resources that can provide means for reaching a goal [Yu95].

Contribution analysis can be thought of as a special case of means-end analysis in which means are goals or soft-goals. Such analysis identifies goals that can contribute either positively or negatively to the achievement of other goals (or soft-goals).

On the other hand, decomposition refers to the systematic breakdown of a component into simpler more specific components. Therefore, during goal modelling, goal decomposition refers to the systematic breakdown of an actor's goals (called root goals) into simpler, more specific sub-goals that may be used to generate tasks, whereas during task modelling, task decomposition results in the decomposition of a root task to sub-tasks. In Tropos, AND/OR decomposition allows developers to consider alternatives when decomposing the goals/tasks of an actor into sub-goals/sub-tasks. Whereas AND decomposition means all the sub-goals/sub-tasks must be achieved for the root goal/task to be achieved, OR decomposition means that the achievement of one of the sub-goals/sub-tasks leads to the achievement of the root goal/task.

Capability modelling [Bre02b] takes place during the latest steps of the architectural design and it involves the identification of capabilities for each of the actors of the system according to the goals, tasks and dependencies of each actor. "Individual" capabilities are assigned to the actors of the system to enable them to define, choose and execute tasks for achieving their goals together with "social" capabilities that allow actors to manage dependencies with the other actors. Capabilities can be identified by analysing the dependency relationships of the actors. In particular each dependency relationship can give place to one or more capabilities triggered by external events [Bre02b]. When the agents of the system have been identified, the capabilities corresponding to each of these agents are furthered specified.

Agents assignment [Bre02b] takes place during the last step of the architectural design and it involves the identification of a set of agent types and the assignment on each one of those of one or more capabilities. This process is not unique and it depends on the analysis that takes place during the previous steps of the architectural design as well as the perspective of the developer for the system in terms of agents. For example, developers might decide to assign one agent for every actor of the system identified in the previous steps of the analysis, or they might assign two agents to a particular actor.

Graphical representations of the models obtained following the above-mentioned activities are given through actor, goal, capability, plan and agent interaction diagrams.

A graphical representation of the model obtained following actor and dependency modelling is illustrated with the aid of an **actor diagram** [Bre02b]. In such a diagram, actors (graphically represented as circles[9]) are modelled together with their goals (represented as ovals) and soft-goals (represented as bubbles) and their dependencies (represented as links between the actors indicating the dependum). An example of an actor diagram is given in Figure 3-2.

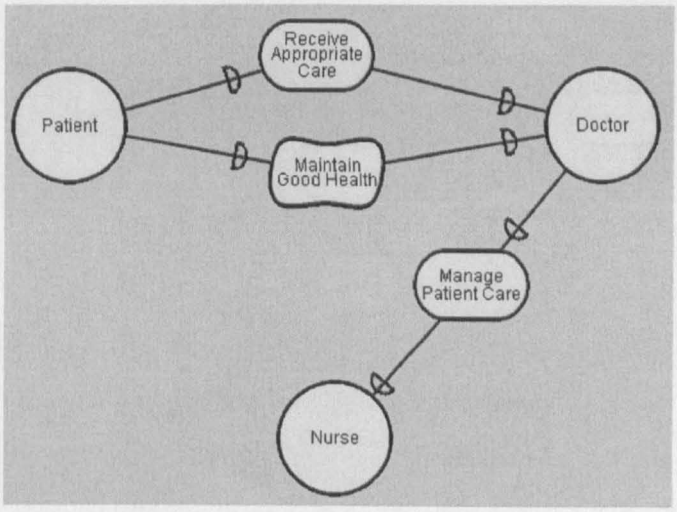


Figure 3-2: An example of an actor diagram

In this example three actors, Patient, Doctor and Nurse are modelled together with some of their dependencies. For example, the Patient depends on the Doctor to

[9] For a reminder of the graphical representation of the Tropos concepts please refer to Figure 3-1.

achieve the goal Receive Appropriate Care whereas the Doctor depends on the Nurse to achieve the goal Manage Patient Care. Moreover, the Patient actor depends on the Doctor to Maintain Good Health. However, maintaining good health is realised differently by different patients. In other words, there are no-clear criteria on the definition of good health and as a result this dependency is modelled as a soft-goal dependency.

Additionally to actor diagrams, Tropos defines **goal diagrams** to represent the models resulting from goal, soft-goal and task modelling activities. In a goal diagram, each actor is represented as a dashed-line balloon within which the actor's goals and dependencies are analysed. The nodes of the diagram represent goals, soft-goals, and/or tasks whereas the links identify the different kinds of relationships between those nodes. Moreover, these links can be connected with external dependencies (identified in the actor diagram) when the reasoning of the analysis goes beyond the actor's boundary [Yu95]. Figure 3-3 shows a partial goal diagram for the Doctor actor presented in the previous example.

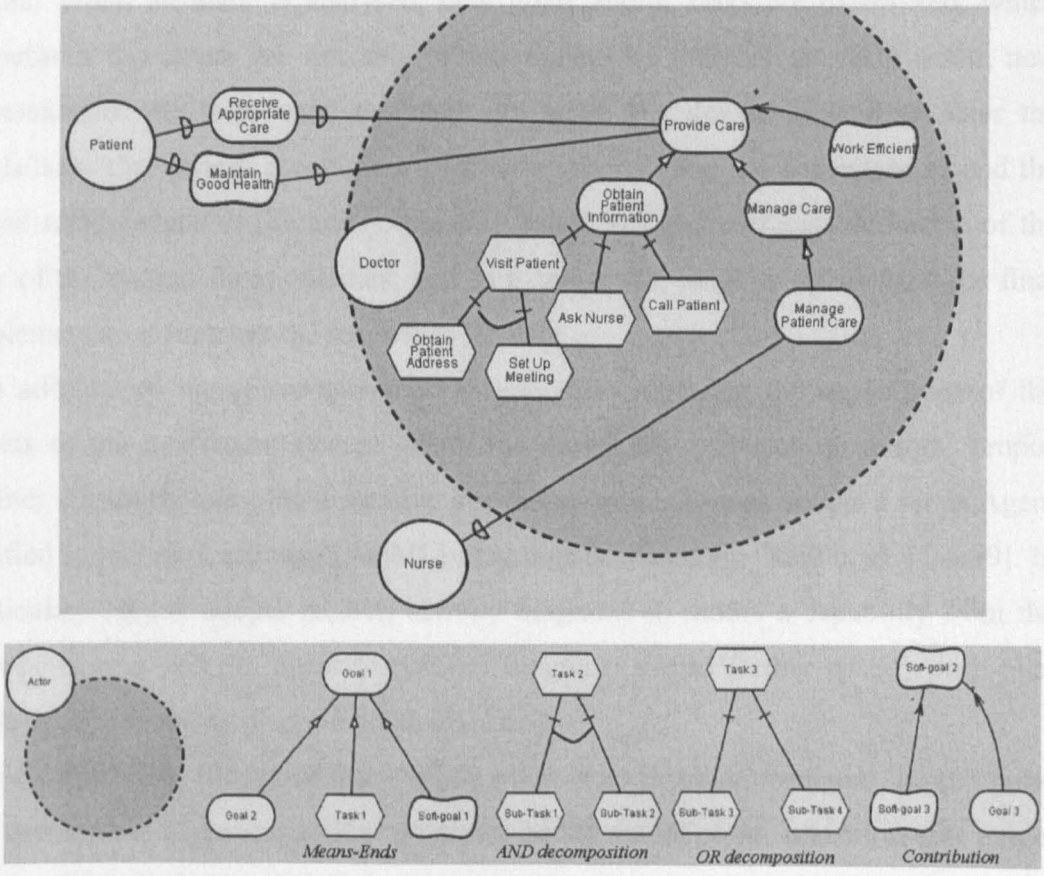


Figure 3-3: An example of a goal diagram

The main goal of the Doctor is to Provide Care. This can be achieved (amongst other goals) by obtaining patient information and manage the patient's care. To achieve the first goal, the doctor can visit the patient, ask the nurse or call the patient. In order for the Doctor to visit the Patient, the patient's address must be obtained and also a meeting must be set up. Moreover, the Provide Care goal of the Doctor receives a positive contribution from the Work Efficient soft-goal. In other words, the more efficient the Doctor works, the better care they will provide. On the other hand, one of the means of achieving the Manage Care goal is to manage the patient's care plan. However, for this goal the Doctor depends on the Nurse.

Apart from analysing the internal goals/tasks of an actor, goal diagrams allow developers to introduce new dependencies between the actors according to the goals/tasks derived from the internal analysis that takes place in each actor. This activity is a common task in Tropos and it is very important since it helps to identify clearly the relationships between the actors and also indicates how the analysis of the goals of one actor can influence the dependencies between this actor and any other actors. When an actor is analysed, new goals and/or tasks are discovered, which sometimes the actors are not able to accomplish by themselves. As a result, new dependencies are introduced to enable an actor to delegate to another actor the goals/tasks that cannot accomplish on their own. Refining the dependencies and the social relationships of the actors this way, leads to a more precise definition of the *why* of the system functionalities, and as a last result, helps to verify how the final implementation matches the real needs [Per01].

In addition to the above-presented diagrams, to represent the capabilities of the agents of the multiagent system, identified during the architectural design, Tropos defines capability and plan diagrams. For this purpose, Tropos adopts a set of Agent Unified Modeling Language (AUML) diagrams proposed by Odell et al. [Ode99]. In particular, Tropos adopts AUML activity diagrams to model a capability from the viewpoint of a specific agent (capability diagram) and to further specify each plan node of the capability diagram (plan diagram).

In a **capability diagram**, the starting point is an external event and the end point the termination of the capability. Activity nodes model plans, transition arcs model events and beliefs are modelled as objects. An example of a capability diagram is

given in Figure 3-4. The capability Receive Service Request of a Receiver Agent (RA) is triggered by an external event (EE). According to this event the Receiver Agent (RA) receives a Service Request from the Sender Agent (SA). The first plan of the capability is for the RA to evaluate the Service Request. If the Service Request is valid an internal event (IE) triggers the Accept Service Request plan and then the termination of the capability, whereas if the Service Request is invalid the Reject Service Request plan is activated and then the capability ends.

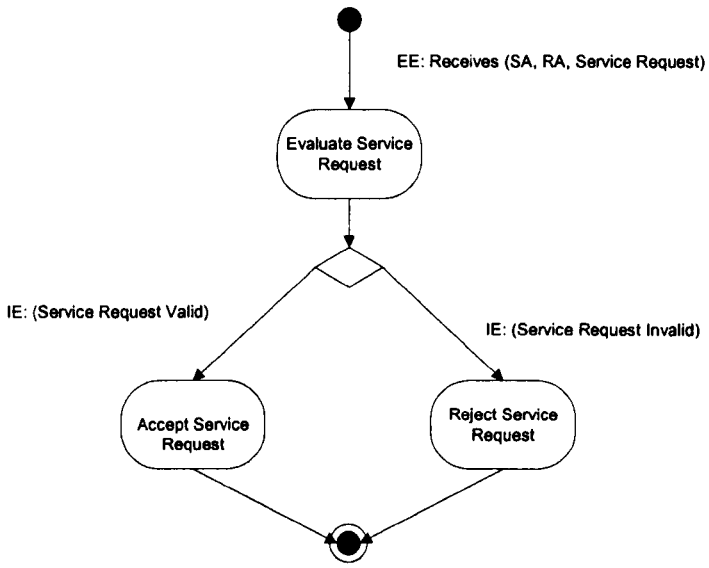


Figure 3-4: An example of a capability diagram

The starting point of a **plan diagram** is the initiation of a plan and the end point is the termination of the plan. The different actions required by the plan are modelled (as activity nodes) together with the transitions (modelled as arcs) from one action to a subsequent one. Consider, for example, the plan Evaluate Service Request presented in Figure 3-5.

The Receiver Agent receives the Service Request from the Sender Agent. To evaluate the Service Request the Receiver Agent first reads it. If the Service Request is readable the Receiver Agent continues by checking its integrity, otherwise the plan is terminated (Fail To Read Service Request). If the integrity check fails, the Receiver Agent considers the Service Request invalid (this is the internal event (IE) identified in Figure 3-4), otherwise the Service Request is valid.

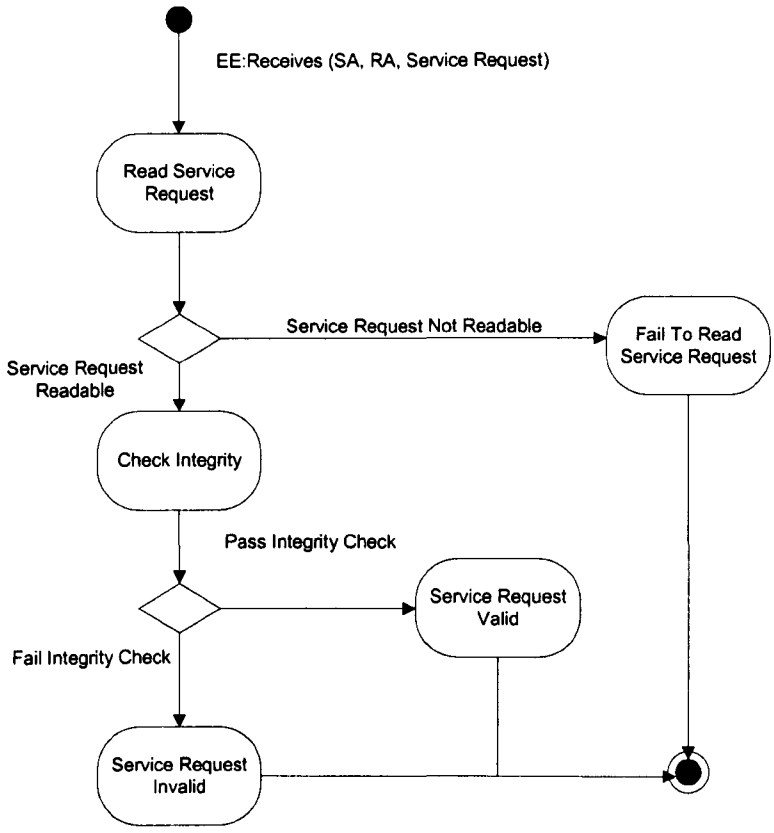


Figure 3-5: An example of a plan diagram

In addition to capability and plan diagrams, Tropos adopts AUML sequence diagrams [Ode99] to model the interactions between the agents of the system. This kind of diagram, which in Tropos is known as **agent interaction diagram**, captures the structural patterns of interactions between the agents of the system by emphasizing the chronological sequence of communications. As an example (see Figure 3-6) consider the sequence of the interactions between the Sender Agent and the Receiver Agent. First the Sender Agent sends the Service Request. Then the Receiver Agent replies with an acceptance of the request or a rejection of the request.

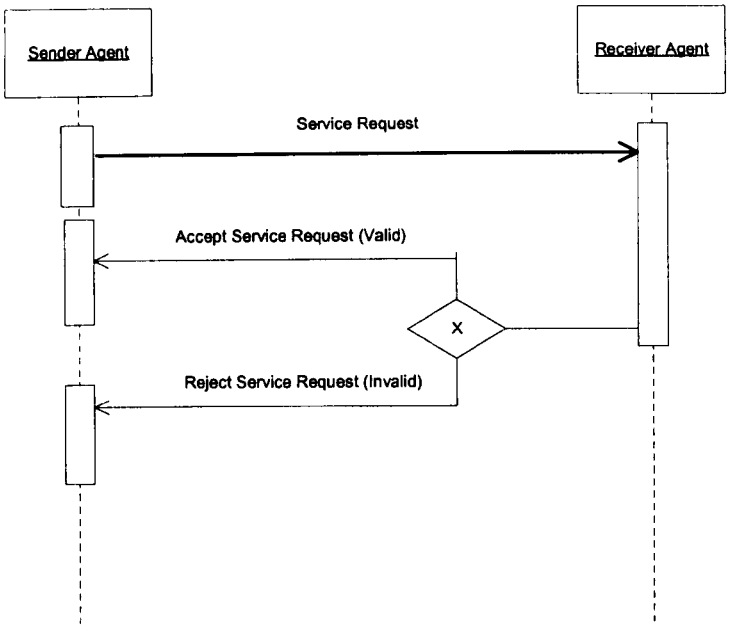


Figure 3-6: An example of an agent interaction diagram

3.6 A SET OF TRANSFORMATIONS

Different ways of visualising an actor and/or a goal diagram can be introduced [Bre02a]. This is due to the fact that different developers have different perspectives of a multiagent system and its environment. Therefore, Tropos introduces a set of transformations, which help developers to refine an initial Tropos model to a final one. Three different categories of transformations are defined: goal, soft-goal and actor.

Goal transformations are divided into four sub-categories [Bre02a]. Goal decomposition, which allows for the decomposition of a goal into AND/OR sub-goals; precondition goal, which allow to list a set of necessary (but not sufficient) preconditions in terms of other goals; goal delegation, which allows to express the assignment or a change of responsibility in goal fulfilment; and goal generalisation, which allow the introduction of an ISA hierarchy[10] among two goals.

Soft-goal transformations [Bre02a] allow developers to perform soft-goal analysis and are very similar to the goal transformations. The only difference is the

[10] An ISA hierarchy denotes a generalisation relationship between two entities. For example if entity A *ISA* entity B then B is a generic entity and A is a specialisation of it.

lack of a precondition transformation and the addition of a contribution transformation. Contribution transformations [Bre02a] allow developers to specify whether a goal or soft-goal contributes to some other soft-goal or whether there is a goal or soft-goal that contributes positively or negatively to the soft-goal satisficement.

Tropos provides two types of **actor transformations** [Bre02a], actor aggregation and actor generalisation. Actor aggregation [Bre02a] involves the recognition of different actors as part of an organisation or a system, whereas actor generalisation [Bre02a] allows developers to introduce taxonomic structure among actor types. As an example consider a medical system that contains the National Health Service (NHS) as an actor. Aggregating NHS means the actor is decomposed into different departments and responsibility for different NHS goals is delegated into those departments. On the other hand, NHS could be classified as Government Institution (generalisation-*ISA* hierarchy). Therefore, the NHS actor could inherit goals that could be identified on a previous analysis regarding Government Institutions.

Moreover, the above-mentioned transformations are formally defined by adopting notions of Graph Transformation systems [Bre02]. In particular, a set of rules and an algorithm have been developed [Bre02] that allow developers to perform a precise inspection of the models development.

3.7 AN EXAMPLE OF USING TROPOS

In this section, the Tropos methodology is illustrated with the aid of an example. In this example, a simplified version of an agent based system to deliver the single assessment process [Mou03c] is considered.

3.7.1 Early requirements analysis stage

As it was mentioned in section 3.3, the first step in the Tropos methodology is to represent the system's domain actors and the dependencies between them with the aid of the actor diagram. In the presented example five actors are taken into account:

- Older Person: The older person actor represents patients aged 65 or above, who wish to receive appropriate health and social care.

- Professional: The professional actor represents any primary care professional, such as general medical practitioners, nurses and social workers, involved in the older persons' care.
- DoH: The DoH actor represents the English Department of Health.
- Benefits Agency: The benefits agency actor represents a financial agency that helps older persons financially.
- R&D Agency: The R&D Agency actor represents a research and development agency interested in obtaining older person clinical data to perform analysis.

The actor diagram for the above actors is shown in Figure 3-7. The Older Person actor has a main goal to Receive Appropriate Care and a soft-goal to Maintain Good Health. However, the Older Person cannot achieve these two goals on their own so they depend on the Professional actor to accomplish them. In addition, the Older Person depends on the Benefits Agency to Receive Financial Support. On the other hand, the Professional actor depends on the Older Person to Obtain Older Person Information and on the Department of Health (DoH) to help them Provide Services to Older Person.

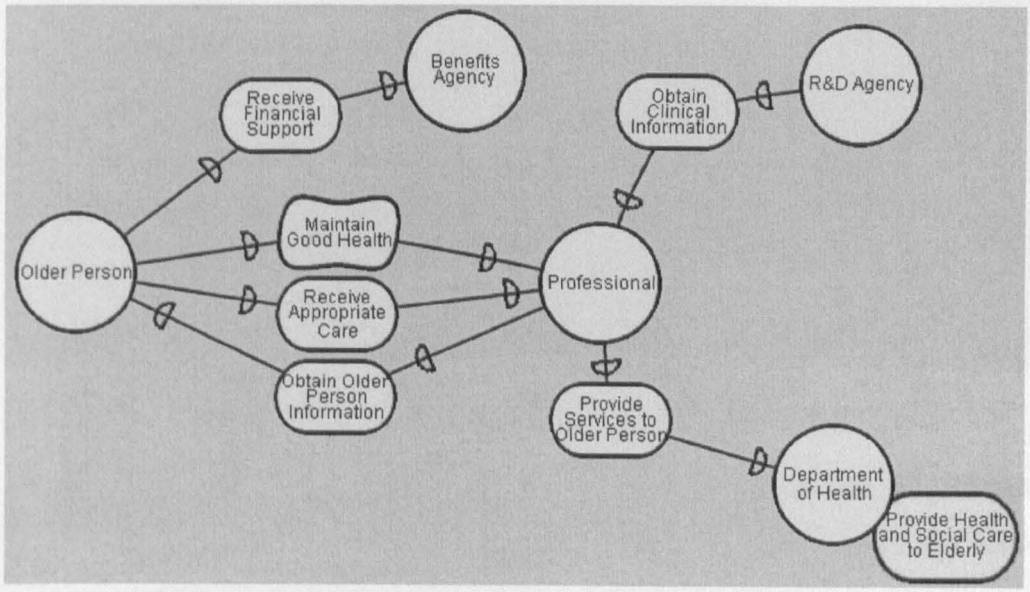


Figure 3-7: The actor diagram for the given example

One of the main goals of the R&D Agency is to Obtain Clinical Information in order to perform tests and research. To get this information, the R&D Agency depends on the Professional. The DoH actor has a main goal to Provide Health

and Social Care to Elderly. However, differently than the other presented actors, the Department of Health is able to accomplish this goal without help from any of the other actors (this is the reason the goal is attached to the DoH, Figure 3-7, and does not involve any dependency).

When the actors, their goals and the dependencies between them have been identified, the next step of the early requirements analysis stage involves in depth analysis of each of the actors. As mentioned earlier (section 3.5), for this purpose Tropos employs goal diagrams.

A part of the goal diagram for the Older Person actor is shown in Figure 3-8.

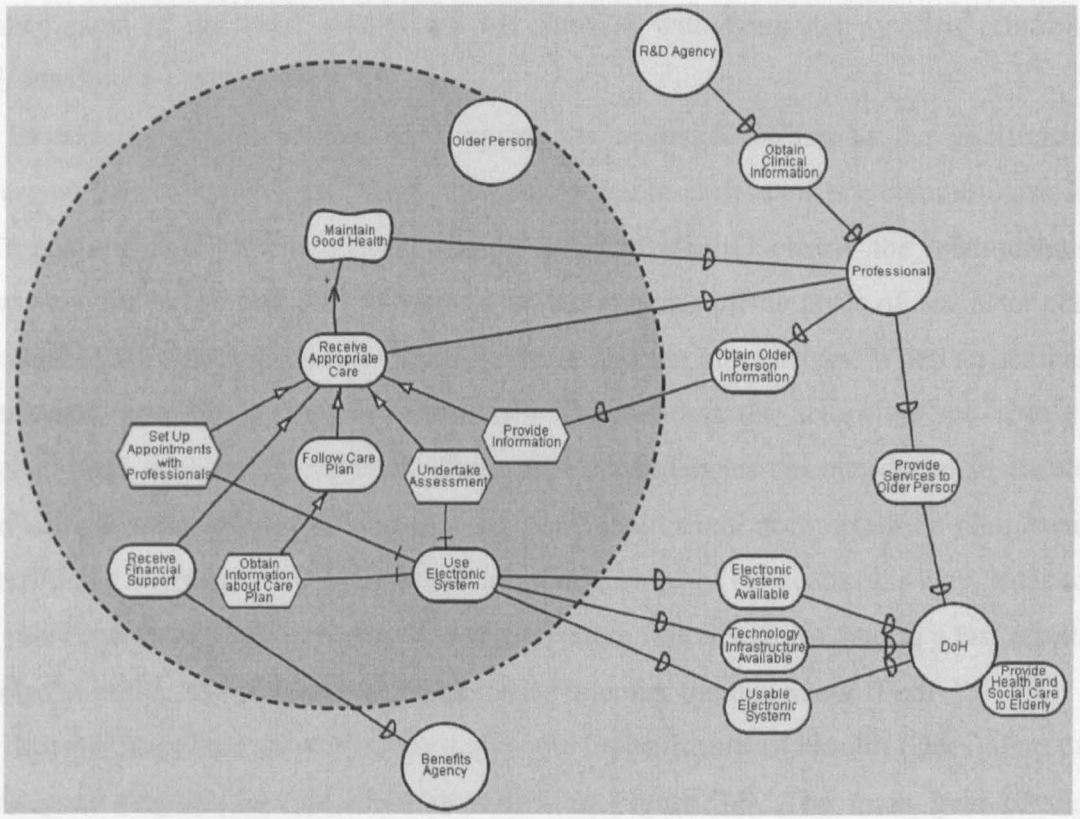


Figure 3-8: Part of the goal diagram for the Older Person

As mentioned, the main goals of the Older Person actor are to Receive Appropriate Care, and to Maintain Good Health. For these goals the Older Person depends on the Professional. However, the satisfaction of the Receive Appropriate Care goal, does not only depends on the Professional, but also on the Older Person. To accomplish the Receive Appropriate Care, the Older Person must perform the tasks Set Up Appointments with Professionals, Undertake Assessment, and Provide Information. Moreover, the Older Person must satisfy

the Follow Care Plan goal. To achieve this goal the Older Person needs to obtain information about the given care plan (Obtain Information about Care Plan task).

To Set Up Appointments with Professionals, Undertake Assessments and Obtain Information About the Care Plan, the Older Person must use an electronic system. This introduces three more dependencies of the Older Person to the Department of Health. For the Older Person to use the electronic system, the DoH must make the system available (Electronic System Available goal), make available the technology infrastructure that the system will be deployed (Technology Infrastructure Available goal), and also make the system easy to use since most of the older people are not familiar with computer systems (Usable Electronic System goal).

Introducing new dependencies between the actors according to the goals/tasks derived from the internal analysis that takes place in each actor is a common task in Tropos and it is very important since it helps to identify clearly the relationships between the actors and also indicates how the analysis of the goals of one actor can influence the dependencies between this actor and the other actors. When an actor is analysed, new goals are discovered, which sometimes the actors are not able to accomplish them by themselves. Thus, new dependencies are introduced to enable an actor to delegate to another actor the goals that cannot accomplish on their own. Refining the dependencies and the social relationships of the actors this way, leads to a more precise definition of the *why* of the system functionalities, and as a last result, helps to verify how the final implementation matches the real needs [Per01].

Another important actor of the system is the Department of Health (DoH). Part of the goal diagram for the DoH is shown in Figure 3-9. The main goal of the Department of Health is to Provide Health and Social Care to Elderly. To accomplish the Provide Health and Social Care to Elderly goal, the Make Care Person Centred sub-goal has been identified. This is essential for the DoH, since the Older Person is the most important participant of the whole procedure. The Make Care Person Centred sub-goal can be fulfilled by promoting the single assessment process (Promote Single Assessment Process goal) and also by involving elderly in their care (Involve Elderly in their Care goal). The later sub-goal depends on the task Provide Guidelines for Older People to be fulfilled. To

promote the single assessment process, the Department of Health must computerise the process and also provide guidelines to the professionals. Therefore, the goal Promote Single Assessment Process is realised by the fulfilment of the Provide Guidelines for Professionals task. This is further decomposed into four sub-tasks: Provide Guidelines for General Practitioners (GPs), Provide Guidelines for Social Workers, Provide Guidelines for Nurses and Provide Guidelines for Other Professionals.

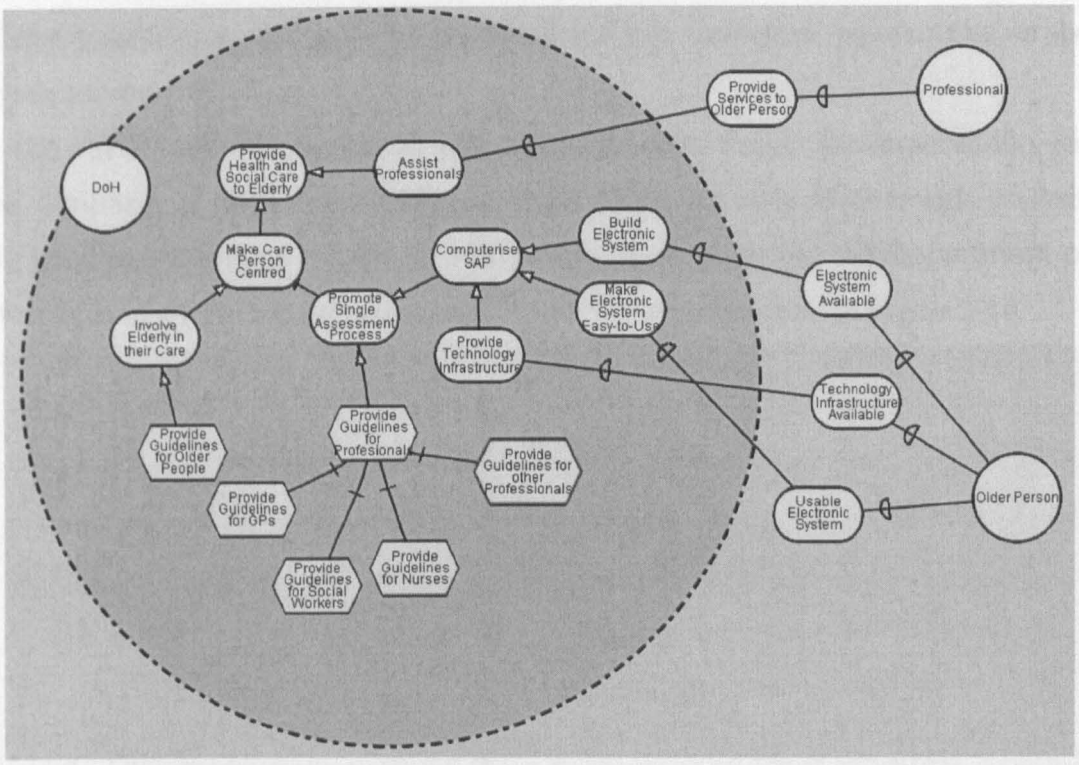


Figure 3-9: Part of the goal diagram for the Department of Health

In addition, to help professionals to Provide Services to Older Person, the Department of Health must fulfil the Assist Professionals goal. To accomplish this goal the sub goal Computerise SAP *(*single assessment process*)* has been identified. Computerising the single assessment process will help health and social care professionals to automate some procedures required while caring for the Older Person and therefore help to Provide Services to Older Person. To accomplish the Computerise SAP sub goal, Technology Infrastructure must be provided, the electronic system must be available (Build Electronic System goal) and also the system must be usable (Make Electronic System Easy-to-Use goal).

3.7.2 Late requirements analysis stage

During the early requirements analysis, the development of an electronic system was identified as one of the main goals of the Department of Health. During the late requirements analysis this system, named the electronic single assessment process (eSAP) system hereafter, is described within its operation environment, along with relevant functions and qualities. The system is presented as one or more actors, who have a number of dependencies with the other actors of the organization. These dependencies define all the functional and non-functional requirements for the system-to-be.

The eSAP system is introduced as another actor that receives the responsibility for the fulfilment of some of the goals identified during the early requirements analysis for the Department of Health. In other words, some goals that the Department of Health cannot fulfil are delegated to the eSAP System as shown in Figure 3-10.

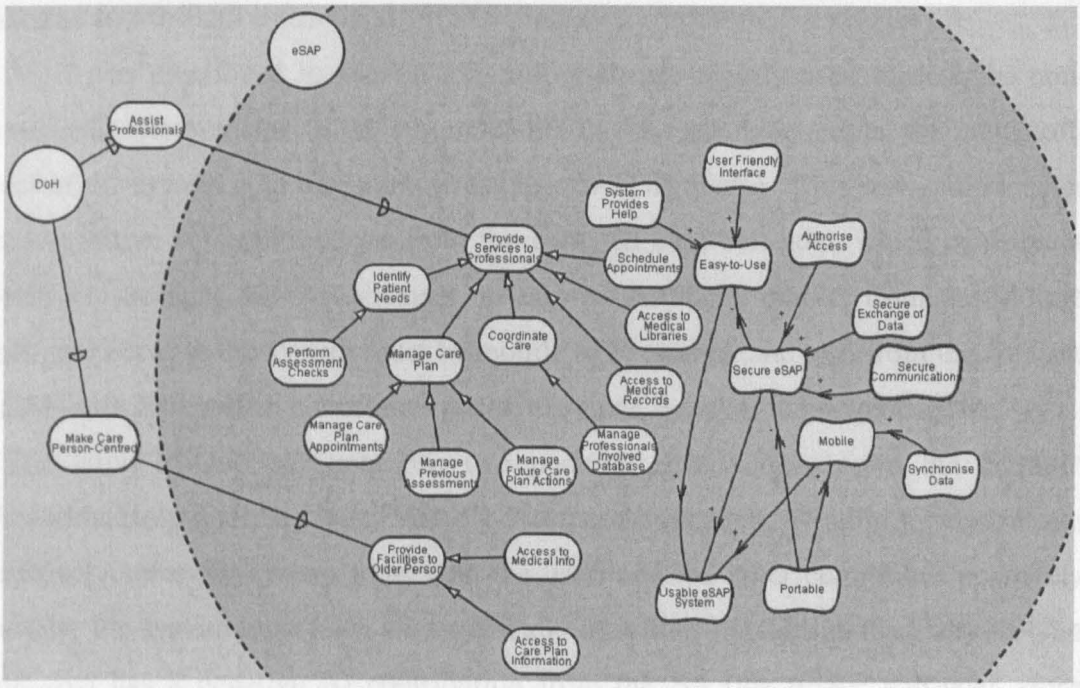


Figure 3-10: Part of the goal diagram for the eSAP

The Department of Health depends on the electronic single assessment process (eSAP) actor to fulfil its two main sub-goals (Assist Professionals and Make Care Person-Centred). To guarantee the satisfaction of these dependencies, the eSAP must Provide Services to Professionals and Provide Facilities to Older Person. With the aid of means-end analysis (section 3.5) it has been identified that

for the eSAP system to fulfil the Provide Services to Professionals goal (end), the following sub-goals (means) must be accomplished: Identify Patient Needs, Manage Care Plan, Coordinate Care, Access to Medical Records, Access to Medical Libraries, and Schedule Appointments. Each of those sub-goals can be furthered analysed employing means-end analysis. For example, the Manage Care Plan can be accomplished with the fulfilment of the Manage Care Plan Appointments, Manage Previous Assessments and Manage Future Care Plan Actions sub-goals.

Another important goal of the eSAP is to Provide Facilities to Older Person. To achieve this goal the eSAP system must allow older people to be actively involved in their care by providing facilities. Thus, the Make Care Person-Centred goal is fulfilled with the achievement of the Provide Facilities to Older Person goal. This is decomposed into two further goals Access to Care Plan Information and Access to Medical Info.

As it was mentioned in section 3.2, soft-goals are mainly used to describe non-functional requirements of the system-to-be. In the running example, the main soft-goal of the system is to be usable (Usable eSAP System). This soft-goal receives three positive (+) contributions from the Easy-to-Use soft-goal, which contributes positively because the system must be easy-to-use to be usable, from the Mobile soft-goal because the system must be mobile to be usable, and also from the Secure eSAP soft-goal, which contributes positively since it makes the system secure.

The Easy-to-Use soft-goal has two positive contributions from the System Provides Help and the User Friendly Interface soft-goals. The former contributes positively since the system must help the user, and the latter contributes positively because the system must have a user-friendly interface. In addition the Easy-to-Use soft-goal has a negative (-) contribution from the Secure eSAP soft-goal, since usually trying to make the system secure makes it more difficult to use.

The Mobile soft-goal accepts two positive contributions from the Portable and the Synchronise Data soft-goals. The former contributes positively because the system must be portable to be mobile, and the latter because the system must be able to synchronise data in order to be mobile.

Furthermore, the Secure eSAP soft-goal receives three positive contributions. The first positive contribution comes from the Authorise Access soft-goal, which contributes positively because the system must be able to Authorise Access to be secure. The other two positive contributions come from the Secure Exchange of Data and the Secure Communications soft-goals. The former acts positively because the exchange of data must be secured, and the latter because any communication of the system must be secure. In addition, the Secure eSAP soft-goal has a negative contribution from the Portable soft-goal because a portable system is more difficult to secure.

As it can be seen from the analysis presented in this section, the late requirements analysis stage follows the same analysis techniques used in the early requirements analysis. The main difference is the idea of introducing the system as another actor. Such an approach is very important and provides advantages since it helps to identify clearly the relationships and the dependencies between the system and the environment that the system will be situated. Medical information systems, such as the electronic single assessment process system, more often are introduced to environments in which non or very little computer expertise is found. Defining clearly the roles and the dependencies of the actors and the system helps to identify the functional and non-functional requirements of the system-to-be according to the real needs of the actors. Also, analysing the system itself within its operational environment helps to delegate responsibility for the achievement of goals to the system and also identify new dependencies between the system and the other actors. This leads to the definition of functional and non-functional requirements for the system, which would be very difficult to identify otherwise. In addition, the way the system is analysed within the late requirements stage, provides developers with the ability to consider different alternatives for satisfying the system's goals and decide, by checking for example if the alternative contributes positively or negatively to the other goals of the system, which of these alternatives is the best solution.

3.7.3 Architectural design stage

When the system goals and soft-goals have been identified, the next step of the development cycle involves the definition of the system's global architecture in

terms of subsystems (actors) interconnected through data and control flows (dependencies).

As mentioned in section 3.3 the first step of the architectural design stage is the identification of actors to take responsibility to fulfil one or more goals of the system and to contribute positively to the fulfilment of some non-functional requirements.

Figure 3-11 shows a partial decomposition of the eSAP actor into sub-actors that have been delegated the goals of the system[11]. The eSAP system depends on the Coordinator Manager to coordinate the care of the older people, on the Medical Library Manager to Provide Access to Medical Libraries, on the Medical Records Manager to Provide Access to Medical Records, on the Appointments Manager to Schedule Appointments, on the Care Plan Manager to manage the care plans, on the Needs Identifier Manager to identify the needs of the patients, and on the Security Manager to fulfil the Secure eSAP System goal.

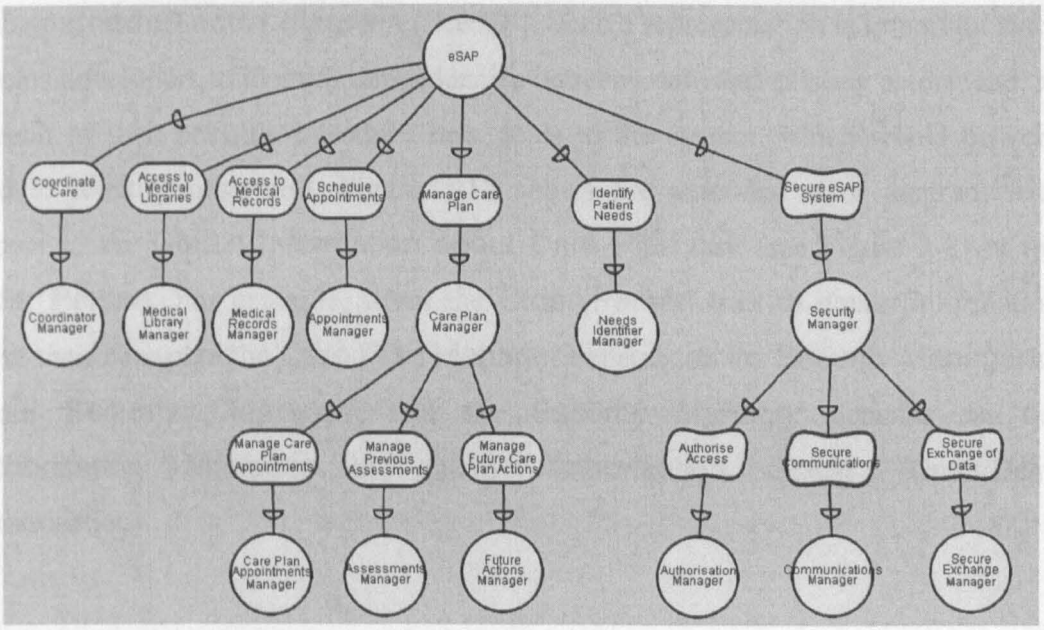


Figure 3-11: Partial decomposition of the eSAP actor

These newly introduced sub-actors can be furthered decomposed as shown in Figure 3-11 to provide more details about the system and allow developers to explicitly define the actors of the system. For example, the Care Plan Manager

[11] In this figure only a partial decomposition is illustrated (not all the goals of the eSAP have been delegated to sub-actors).

depends on the Care Plan Appointments Manager to Manage Care Plan Appointments, on the Assessments Manager to Manage Previous Assessments and on the Future Actions Manager to Manage the Future Actions required by the care plan. Furthermore, the Security Manager depends on the Authorisation Manager to Authorise Access to the system, on the Communications Manager to Secure Communications and on the Secure Exchange Manager to provide security during the exchange of data.

Decomposing the system to sub-systems (sub-actors) and delegate system responsibilities (goals) to those actors help to define more explicitly the system. As argued by Jennings and Wooldridge [Jen99] "*Decomposition helps tackle complexity because it limits the designer's scope: at any given instant only a portion of the problem needs to be considered*".

New actors and their dependencies with the other actors are presented with the aid of the **extended actor diagram** [Bre02b]. Such a representation is important since it helps developers to identify dependencies between new and existing actors, and, as a result of this, possibly introduce new goals to the system, which would be very hard to identify otherwise. Figure 3-12 shows the extended actor diagram with respect to the Obtain Information about Care Plan task (see Figure 3-8) of the Older Person. For example, when the Older Person tries to obtain information about their care plan the Care Plan Manager depends on the Security Manager to obtain Security Clearance, and the Security Manager depends on the Authorisation Manager to obtain Authorisation Status (Grant/Deny Authorisation).

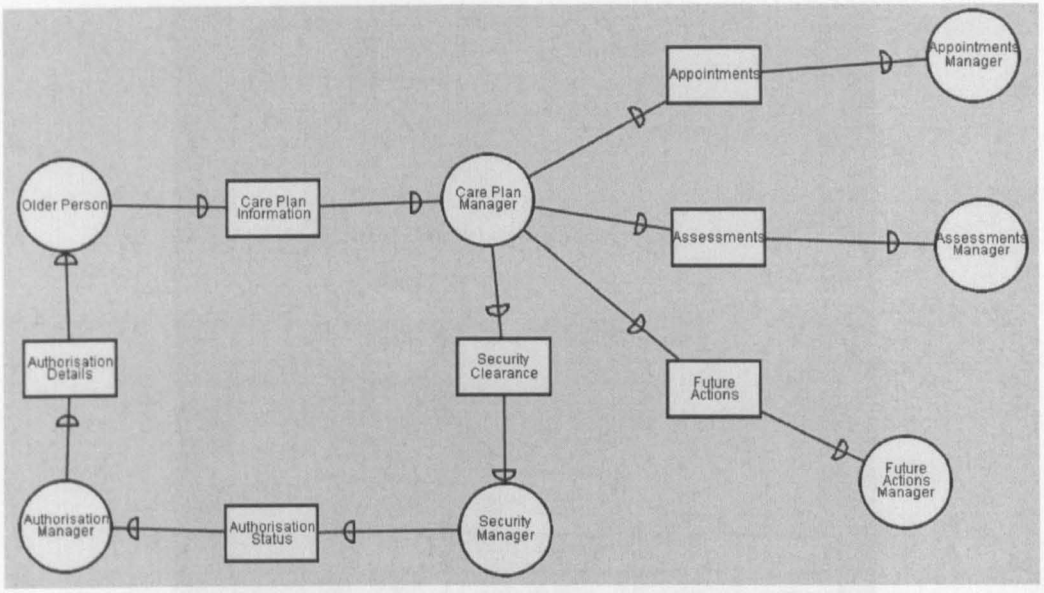


Figure 3-12: Part of the extended actor diagram with respect to the *Obtain Information about Care Plan* task of the Older Person

On the other hand, the Authorisation Manager depends on the Older Person to provide their Authorisation Details.

The actors introduced in the extended actors diagram can be furthered decomposed with respect to their goals and tasks. For example, Figure 3-13 shows a partial decomposition of the Authorisation Manager actor into two sub-actors, the Authorisation Granter and the Authorisation Checker. The former is responsible for checking the Authorisation Data and the Authorisation Privileges and provide (or deny) Authorisation Clearance, and the latter is responsible for checking the user's (in this example the Older Person) Authorisation Details and provide the Authorisation Granter with the Authorisation Data and the Authorisation Privileges of each user.

The architectural design also involves the capabilities identification sub-stage, in which the capabilities needed by each actor to fulfil their goals and tasks are modelled. The extended actor diagram is used to identify the capabilities, since each dependency relationship can give place to one or more capabilities triggered by external events.

For example the resource Authorisation Privileges (modelled in Figure 3-13) calls for the capability Obtain Authorisation Privileges for the Authorisation Granter actor and Provide Authorisation Privileges for the Authorisation Checker actor.

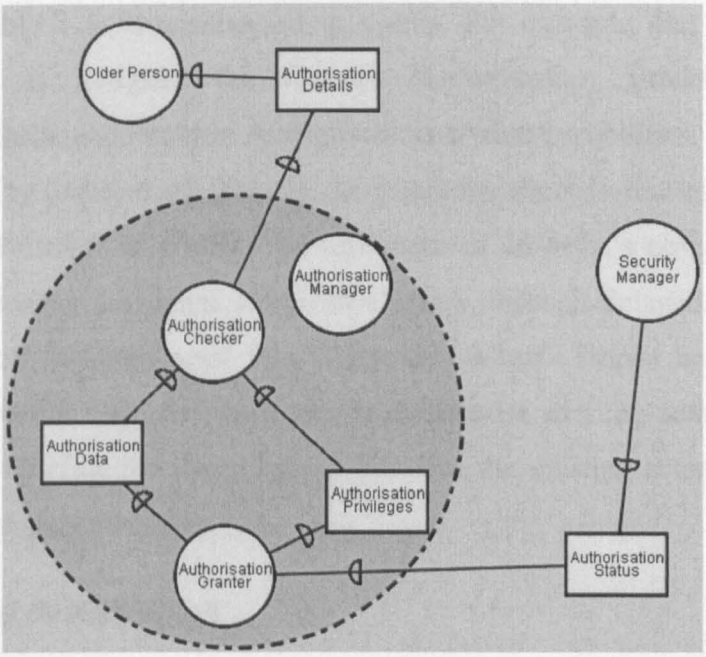


Figure 3-13: Extended diagram with respect to the Authorisation Manager

Later on the detailed design, each agent's capabilities are further specified and then coded during the implementation phase. Table 3-1 reports the actors of Figure 3-13 and their capabilities as derived from the dependencies that exist between them.

Table 3-1: Actors and their capabilities with respect to Figure 3-13

Actor	Capability
Security Manager	Obtain Authorisation Status
Authorisation Granter	Obtain Authorisation Privileges
	Obtain Authorisation Data
	Provide Authorisation Status
Authorisation Checker	Provide Authorisation Data
	Provide Authorisation Privileges
	Obtain Authorisation Details
Older Person	Provide Authorisation Details

The last step of the architectural design is the agents' assignment. During this step a set of agents are defined and each agent is assigned one or more different capabilities identified in the previous step. In the presented example, it was decided (for reasons of simplicity) to allocate capabilities corresponding to each actor

identified in Table 3-1, to corresponding agents. For example, the Authorisation Granter agent is assigned the Obtain Authorisation Privileges, Obtain Authorisation Data, and Provide Authorisation Status capabilities.

As mentioned by Castro et al. [Cas01], an interesting decision that comes up during the architectural design is whether the fulfilment of an actor's obligations will be accomplished through assistance from other actors, through delegation, or through decomposition of the main actor into component actors. Tropos helps developers towards this direction, by allowing them to decompose existing actors, and/or add new actors and redefine the dependencies between the existing actors and the new introduced actors and sub-actors.

3.7.4 Detailed design stage

As mentioned in section 3.3, detailed design stage aims at specifying agent capabilities, plans, and interactions and it is intended to introduce additional detail for each architectural component of the system. For this reason Tropos employs capability, plan and agent interaction diagrams (for a reminder see section 3.5). For example, the Obtain Authorisation Status capability (see Table 3-1) of the security manager agent is illustrated in Figure 3-14.

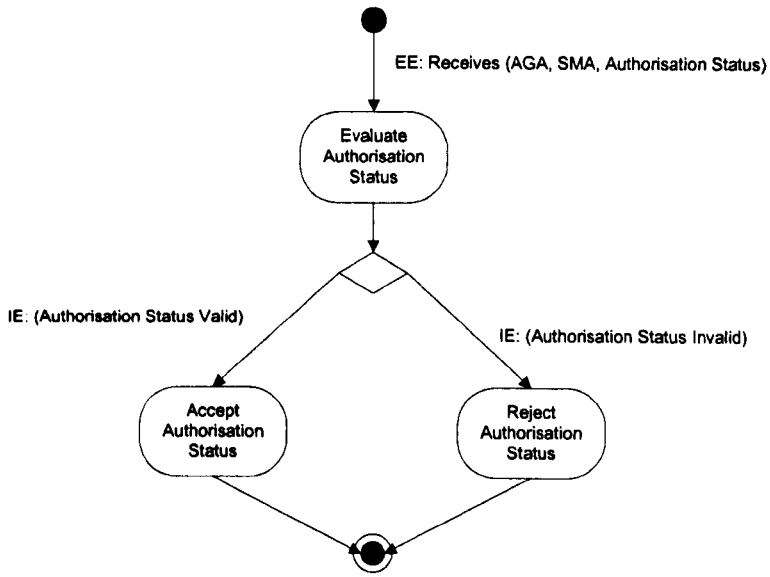


Figure 3-14: Capability diagram for the authorisation status capability

The Security Manager Agent (SMA) receives (external event – EE) the Authorisation Status from the Authorisation Granter Agent (AGA), it evaluates the Authorisation Status and either accepts it or rejects it.

Moreover, each capability depicted on the diagram can be furthered analysed with the aid of the plan diagram. Figure 3-15 illustrates the plan diagram for the Evaluate Authorisation Status plan belonging to the capability depicted in the diagram of Figure 3-14. The plan is activated with the receipt of the Authorisation Status from the Authorisation Granter Agent and it ends by deciding if the Authorisation Status is valid or invalid (In addition the plan can be terminated if Authorisation Status is not readable). The integrity of the Authorisation Status is checked. If the check is successful the Authorisation Status is received as valid, else the Authorisation Status is considered invalid from the Security Manager Agent.

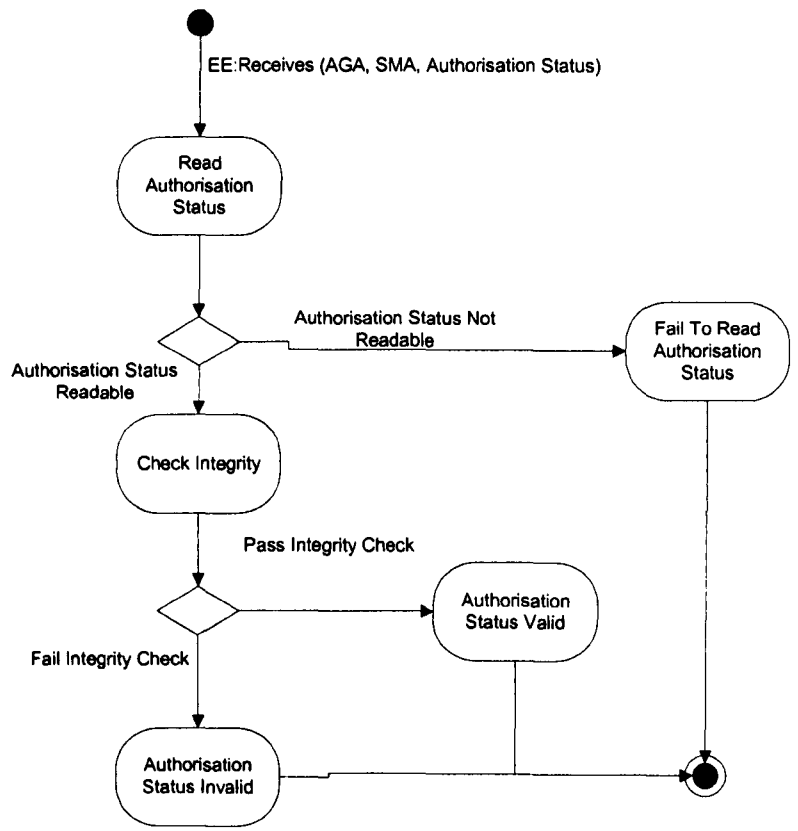


Figure 3-15: Plan diagram for the evaluate authorisation status plan

In addition, an example of an **agent interaction diagram** is shown in Figure 3-16. This diagram illustrates interactions (shown as arrow-lines) between the Security Manager, the Authorisation Granter, the Authorisation Checker, and the Older Person agents (graphically illustrated as rectangles at the top of the

diagram). The Security Manager requests an Authorisation Status from the Authorisation Granter. When the Authorisation Granter receives the request it requests the Authorisation Data and the Authorisation Privileges from the Authorisation Checker. Then the Authorisation Checker sends a request to the Older Person for its Authorisation Details. When the Older Person replies with the Authorisation Details, the Authorisation Checker sends the Authorisation Data and the Authorisation Privileges to the Authorisation Granter, who replies to the Security Manager with the Authorisation Status.

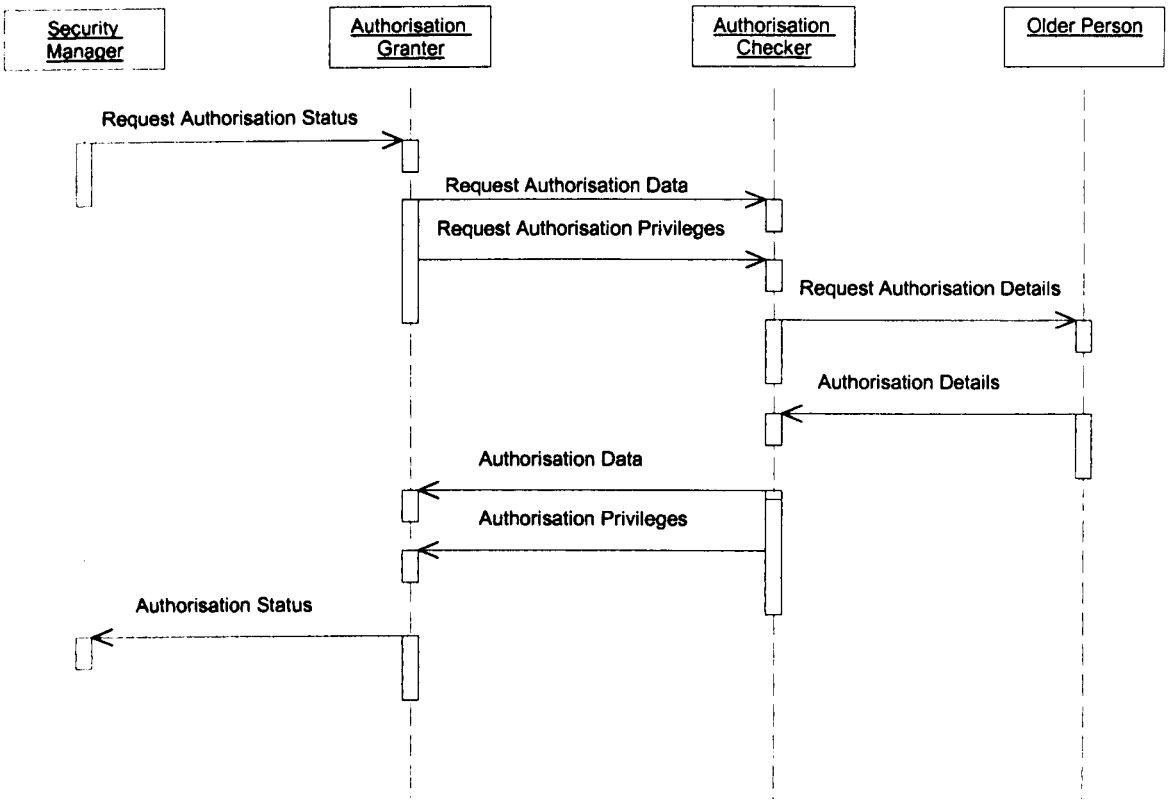


Figure 3-16: Example of an agent interaction diagram

In Tropos the detailed design stage is based on the specifications resulting from the architectural design phase and the reasons for a given element, designed at this level, can be traced back to early requirements analysis, a very important advantage of the methodology.

3.8 LIMITATIONS OF TROPOS WITH RESPECT TO SECURITY MODELLING

The decision of choosing Tropos for the integration of security issues was based, as described in chapter 2, on the potential that Tropos demonstrated, in comparison with other existing methodologies, in being extended with respect to security modelling.

On the other hand, the Tropos methodology demonstrates some limitations with respect to security modelling. This section aims to identify these limitations. The criteria for the evaluation of the Tropos will be based on the requirements identified in the previous chapter, section 2.3.2.2.

3.8.1 Limitations on the concepts of the methodology

As mentioned, the Tropos methodology partially tackles security modelling by allowing developers to capture security requirements, as well as any other non-functional requirements, as soft-goals. The concept of soft-goal is "*used to model quality attributes for which there are no a priori, clear criteria for satisfaction, but are judged by actors as being sufficiently met*" [Yu95]. However, security requirements relate to system's quality attributes, or alternatively may define constraints on the system [Som01, Rom85]. Qualities are properties or characteristics of the system that its stakeholders care about, whereas constraints are restrictions, rules or conditions imposed to the system and unlike qualities are (theoretically) non negotiable. Therefore, although the concept of a soft-goal captures qualities, it fails to adequately capture constraints. However, possible constraints might be imposed on the system representing restrictions (global or for each individual agent). For example, security constraints might be imposed on the system representing restrictions related to its security. Such constraints might affect the analysis and design of the system, by restricting some alternative design solutions, by conflicting with some of the requirements of the system, and also by refining some of the goals of the system or introducing new ones that help the system towards the satisfaction of its requirements.

To further illustrate the need to introduce constraints in the Tropos methodology, consider the actor diagram presented in Figure 3-7. By analysing the actor diagram of this example, it is observed that although the dependencies between the actors are clearly shown, some possible constraints that might be imposed to some of the actors are not present. For example, the Older Person depends on the Benefits Agency to Receive Financial Support but the Older Person most likely introduces a security-related constraint to the Benefits Agency to keep their financial information private. On the other hand, the R&D Agency actor depends on the

Professional actor to Obtain Clinical Information but the Professional might be restricted (for example by the DoH or the Older Person) to provide only anonymous clinical information. In addition, the Older Person might restrict the Professional by imposing a constraint to share medical information only if the older person's consent is obtained.

Therefore, the above-mentioned actors have to achieve their goals while having to satisfy different security constraints imposed to them. By analysing the constraints that actors might impose to each other, developers are able to identify security goals that can be used later in the development process and which (the goals) help towards the identification of the security requirements of the multiagent system. However, currently the Tropos methodology fails to adequately model such constraints, and therefore the modelling of security issues during the development of a multiagent system is restricted.

In addition, the usage of soft-goals to model general non-functional requirements although it allows developers to define together security and other functional and non-functional requirements, it does not help in providing a clear distinction between the security and the other requirements of the system (requirement 6 in section 2.3.2.2). Such a distinction is made even harder by the lack of definition of the Tropos concepts, such as goals, tasks, and dependencies, with security in mind.

3.8.2 Limitations on the Tropos' process of modelling security

In addition to the above limitations regarding the concepts of the Tropos methodology, there are limitations regarding the process of modelling security issues. The current process, of the Tropos methodology, of modelling and reasoning about security issues throughout the whole range of the development stages of multiagent systems is quite ad hoc. Developers are allowed to capture security requirements with the aid of soft-goals, and then propagate them throughout the development stages. Also, the methodology allows developers to (partially[12]) identify conflicts between security and other requirements. However this process is neither clearly nor well guided (requirement 4 in section 2.3.2.2). It is unclear how

[12] Partially because the methodology identifies conflicts only between security requirements captured by soft-goals and not any security constraints that the system could be imposed.

developers can systematically capture security requirements (expressed as soft-goals) and how they can develop a design that successfully meets those requirements in a systematic way (requirement 9 in section 2.3.2.2). For example, it is not defined by the methodology how soft-goals related to security and identified during the analysis process can be transformed to security goals of the system during the design and how these soft-goals can be traced back in the early requirements analysis stage.

In addition, the methodology does not provide any process to allow developers to reason about the consequences of the application of a particular design to their system (requirement 8 in section 2.3.2.2) and also fails to provide a process that allows developers to evaluate the developed security solution (requirement 9 in section 2.3.2.2). Consider for instance the example presented in section 3.7. How can developers know that the proposed design actually meets the security requirements?

Moreover, the methodology assumes developers demonstrate in-depth security knowledge. This is due to the fact that in order to express security requirements as soft-goals, developers have to identify these security requirements. For instance, consider the security analysis of the eSAP system (see Figure 3-10). The security soft-goal (Secure eSAP) receives positive contributions from three soft-goals (Authorise Access, Secure Communications and Secure Exchange of Data). However, currently, the introduction of these soft-goals depends only on the knowledge of security that each developer has and there is no a systematic way to introduce them to the system according to any kind of analysis. For novice-security developers, who lack knowledge of security, this is a very difficult task since the methodology does not provide any particular process to help them to identify such security requirements (requirement 1 in section 2.3.2.2).

In addition, the methodology fails to integrate security modelling during the early requirements analysis stage. For instance in the example presented in section 3.7, security is introduced only on the eSAP system analysis. However, all the actors play an important role with respect to the security of the system and all of them should be analysed with security in mind. Someone might argue that the same way security was (partially) considered during the eSAP analysis, could be considered for all the actors. However, the point here is that Tropos fails to provide a process

that will guide security-novice developers in identifying that such an analysis should take place not only for the eSAP system but for all the actors related.

3.8.3 Discussion with respect to the limitations

From all the above it is concluded that the Tropos methodology does not provide a structured approach towards security modelling (requirement 3 in section 2.3.2.2) and therefore needs to be extended in order to adequately model security issues. Extensions are required to the ontology of the methodology as well as in the development process. Extensions on the ontology should involve the introduction of the concept of constraint and the definition of the current Tropos concepts with security in mind.

An alternative way (than extending the ontology) in modelling security in the Tropos methodology would be to introduce goals (related to security) to the actors without first imposing any constraints. For instance, in the electronic single assessment process example, a goal such as Obtain Older Person Consent could be introduced to the Professional actor without analysing any constraints that could be imposed to this actor. This would be possible, but it would represent a totally ad hoc process, depending only on the experience and the capability of the developer. Therefore, such an approach restricts the use of the methodology only to security expert developers and it would be in contrast with one of the important requirement of a security oriented approach, which is to allow novice security developers to successfully consider security issues during the analysis and the design of a multiagent system. Moreover introducing goals without defining them by taking into account security it makes the distinction between the security and the other requirements of the system extremely difficult.

On the other hand, someone might argue that constraints could be captured as goals. Nevertheless, the concept of a constraint is different from the concept of a goal. A goal represents a desired state of the world, while a constraint represents a condition, rule, or restriction towards the achievement of a goal. Although a goal can be achieved with various ways, a constraint defines a set of restrictions on how the goal will be achieved. For example, the Benefits Agency could have a goal to keep financial information private. However this is not a goal of the Benefits Agency,

the goal is to provide financial support, but rather a restriction imposed in achieving the goal.

Therefore, as derived from the presented discussion, the ontology of the Tropos methodology should be extended to include the concept of constraint (and also define the concept with security in mind) and in addition the Tropos concepts should be defined with security in mind. In addition, extensions to the development process of the methodology are essential to enable a structured security-oriented approach in the development of multiagent systems. In particular extensions regarding the development process should satisfy the requirements identified in the previous sub-section (3.8.2) that currently Tropos fails to meet.

3.9 SUMMARY

Tropos is an agent oriented development methodology based on intentional and social concepts inspired by the early requirements analysis. The architecture and software design models produced in Tropos are intentional in the sense that system components have associated goals that are supposed to fulfil and they are also social in the sense that each component has obligations/expectations (expressed in terms of dependencies) towards/from other components [Cas02].

This chapter provided an overview of the Tropos methodology. The concepts and notations, the stages and the modelling language of the methodology were presented. Furthermore the modelling activities and a set of transformations defined by the Tropos methodology were introduced.

This chapter also provided a critical discussion, evaluation, of the methodology with respect to security modelling. The limitations of the Tropos methodology, as derived from an evaluation against the requirements presented in chapter 2, were identified and a preliminary discussion on the required extensions took place.

The aim of the next two chapters is to introduce those security-oriented extensions and discuss how they can be integrated within the development stages of the Tropos methodology. More specifically, chapter 4 introduces the proposed security concepts and security-oriented modelling activities, whereas chapter 5 describes the proposed security-oriented approach and it explains how the approach can be integrated within the Tropos development stages.

CHAPTER 4 SECURE CONCEPTS AND MODELLING ACTIVITIES

The previous chapter introduced the concepts, the modelling activities and the development process of the Tropos methodology. Furthermore, it identified the limitations of the methodology with respect to security modelling. To overcome those limitations, this research has extended the Tropos methodology to enable it to model security issues during the development process of a multiagent system.

The purpose of this chapter is to describe security-oriented extensions to the concepts and the modelling activities of the Tropos methodology. Section 4.1 outlines how this research approached the issue of integrating security in the Tropos methodology. The newly introduced and the extended concepts are presented in section 4.2, and section 4.3 describes the modelling activities with respect to the security modelling. Finally, section 4.4 summarises the chapter.

4.1 INTEGRATING SECURITY IN THE TROPOS METHODOLOGY

The main challenge when integrating security modelling issues in a development methodology is to provide a security-oriented approach that will allow developers to provide as much effective security as possible, by systematically analysing the security issues of the multiagent system, and successfully integrate such an approach in the development stages of the methodology.

Having this in mind, the extensions provided by this research to the Tropos methodology, in order to accommodate a security-oriented approach during the development of multiagent systems, can be divided into two main categories: (1) extensions related to the ontology and the modelling activities of the methodology; and (2) extensions related to the development process of the methodology.

The first category involves the introduction of new security-related concepts such as security constraints and the definition of current concepts, such as goals, tasks,

resources, capabilities and dependencies, *with* and *without* security in mind. Consider, for example, the difference between a goal and a *secure goal*. The latter representing a goal that specifically affects the security of the system.

The second category involves the development of a security-oriented process and the integration of this process into the development stages of the Tropos methodology. Towards this direction, this research has developed processes that allow developers to identify the security requirements of a multiagent system, to select amongst different architectural styles with respect to the security requirements of the system, to transform a multiagent system's security requirements to design, and to evaluate the security of the system. In addition, these have been successfully integrated within the development stages of the Tropos methodology.

The rest of this chapter focuses on the first category[13]. Therefore, it introduces extensions to the Tropos ontology, by describing the concept of *security constraints* and the definition of existing Tropos concepts with respect to security modelling. In addition, the chapter describes security-related modelling activities involving the presented security concepts.

4.2 THE SECURE CONCEPTS

As derived from the analysis presented in chapter 3, the current ontology of the Tropos methodology fails to adequately model security during the development process of a multiagent system. To enable developers to adequately capture security requirements this research introduces the concept of constraint and it extends it with respect to security. In addition, the Tropos concepts of dependency, goal, task, resource, and capability are also extended with security in mind. This section aims to describe these concepts, which are defined within the Tropos project as secure concepts.

4.2.1 Constraint and security constraint

As discussed in chapter 3, section 3.7, the current ontology of Tropos fails to adequately model security constraints related to the development of multiagent systems. However, before defining the concept of security constraints within the

[13] Extensions related to the development process of the methodology are presented in Chapter 5.

Tropos methodology, the concept of constraint has to be defined within the Tropos context.

Constraints can represent a set of restrictions that do not permit specific actions to be taken or prevent certain objectives from being achieved and more often [Ste95] are integrated in the specification of existing textual descriptions. However, this approach can often lead to misunderstandings and an unclear definition of a constraint and its role in the development process. Consequently, this results in errors in the very early development stages that propagate to the later stages of the development process causing many problems when discovered; if they are discovered.

Therefore, it is important to define constraints, as a separate concept of the Tropos ontology. To this end, the concept of constraint has been defined within the context of this project as follows:

A restriction that can influence the analysis and design of the multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives.

Additionally, to fully integrate the concept of a constraint in the Tropos methodology, this research has extended the metamodel of the Tropos modelling language by introducing the construct for modelling constraints. The portion of the Tropos metamodel concerning the concept of constraint is shown in the Unified Modelling Language (UML) class diagram of Figure 4-1.

A constraint restricts zero or more (0...*) dependencies, goals and/or tasks. Conversely zero or more (0...*) dependencies, goals and/or tasks are restricted by one or more (1...*) constraints. When a constraint is imposed to a goal (or task), two analysis processes are employed: Constraint decomposition, which aims to further decompose the constraint; and goal introduction, which identifies possible goals that the constraint might introduce to the system.

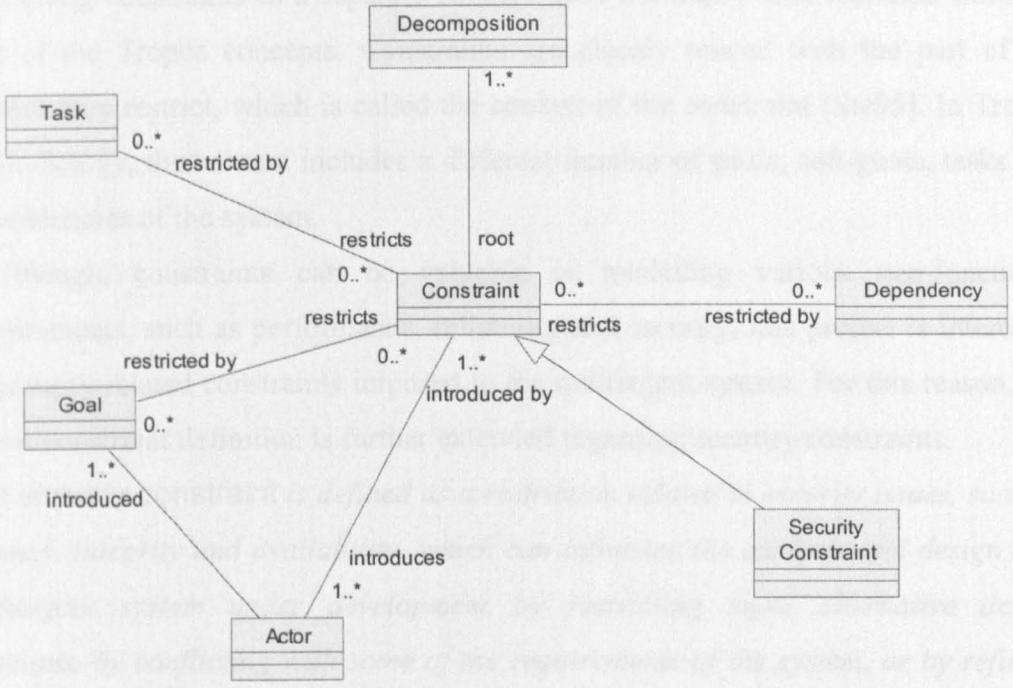


Figure 4-1: UML metamodel for the concept of constraint

Performing these types of analysis, the developer goes from a very high level definition of a constraint to a more detailed and precise definition. In the same time, constraint analysis allows designers to check and refine the goals of an actor according to the imposed constraints, and decide how these goals can be better satisfied.

A constraint can be decomposed into one or more (1…*) sub-constraints. Sub-constraints define more precisely a constraint. The decomposed constraint is called the "root" constraint. However, unlike a goal in which the decomposition provides a set of necessary sub-goals (AND-decomposition) and/or alternatives sub-goals (OR-decomposition), the fulfillment of which has to be considered as necessary and sufficient condition for the fulfillment of the higher goals, a constraint decomposition implies the satisfaction of the root security constraint, if and only if all the sub-constraints are satisfied.

Moreover, constraints can introduce goals to an actor. This is known as goal introduction. The purpose of these goals is to help towards the achievement of the constraint. In other words, during the process of goal introduction, the developer refines the goals of an actor to allow the satisfaction of a constraint.

Defining constraints as a separate concept does not imply their isolation from the rest of the Tropos concepts. Constraints are closely related with the part of the system they restrict, which is called the context of the constraint [Ste95]. In Tropos methodology, the context includes a different number of goals, soft-goals, tasks and dependencies of the system.

Although, constraints can be valuable in modelling various non-functional requirements, such as performance, reliability and security, this project is interested in security-related constraints imposed to the multiagent system. For this reason, the above constraint definition is further extended regarding security constraints.

A security constraint *is defined as a restriction related to security issues, such as privacy, integrity and availability, which can influence the analysis and design of a multiagent system under development by restricting some alternative design solutions, by conflicting with some of the requirements of the system, or by refining some of the system's objectives.*

A *security constraint* contributes to a higher level of abstraction, meaning that security constraints do not represent specific security protocol restrictions[14], which restrict the design with the use of a particular implementation language. This higher level of abstraction allows for a generalised design free of models biased to particular implementation languages. Regarding the constraint metamodel, a security constraint is captured through a specialisation of constraint into the subclass security constraint (see Figure 4-1).

Security constraints can influence the security of the system either positively or negatively. Therefore, this research differentiates between positive and negative security constraints. Positive security constraints contribute positively towards the achievement of the security of the system, whereas negative security constraints might put in danger the security of the system. An example of a positive security constraint could be allow access only to personal information and an example of a negative security constraint could be send information plain text (not encrypted).

[14] Such security restrictions should be specified during the implementation of the system and not during the analysis and design.

Security constraints can be categorised into two main categories, human-imposed or environment-imposed. The first category includes security constraints imposed by the stakeholders or the users. As an example consider a security constraint imposed by one actor to another. The second category involves security constraints imposed by organisations, security policies, laws, rules or regulations. For example consider a security constraint imposed to an actor of a system because of the security policy of the organisation. Security constraints imposed by humans can either positively or negatively contribute towards the security of the system, whereas the security constraints imposed by the environment mainly contribute positively. This is due to the fact that humans can impose constraints related to the security of the system regardless if these constraints help or put in danger the security, whereas security constraints imposed by, for example, security policies aim to help towards the security of the system.

Constraints and security constraints are depicted, as illustrated in Figure 4-2, as clouds within which the description of the (security) constraint is shown. The only difference is an S (Security) within brackets that appears in the beginning of the security constraint description to indicate that the constraint is related to the security of the multiagent system.

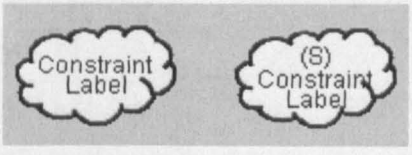


Figure 4-2: Graphical representation of a constraint and a security constraint

4.2.2 Secure dependency

A secure dependency introduces security constraint(s) that must be fulfilled for the dependency to be satisfied. Both the depender and the dependee must agree for the fulfilment of the security constraint in order for the secure dependency to be valid. That means the depender expects from the dependee to satisfy the security constraint(s) and also that the dependee will make an effort to deliver the dependum by satisfying the security constraint(s).

There are three different types of a secure dependency:

- **Dependee Secure Dependency**, in which the depender depends on the dependee and the dependee introduces security constraint(s) for the dependency. The depender must satisfy the security constraints introduced by the dependee in order to help in the achievement of the secure dependency. This type of secure dependency is graphically represented with a security constraint at the side of the depender (see Figure 4-3-a).
- **Depender Secure Dependency**, in which the depender depends on the dependee and the depender introduces security constraint(s) for the dependency. The dependee must satisfy the security constraints introduced by the depender, otherwise the security of the dependency will be in risk. This type of secure dependency is graphically represented with a security constraint at the side of the dependee (see Figure 4-3-b).
- **Double Secure Dependency,** in which the depender depends on the dependee and both the depender and the dependee introduce security constraints for the dependency. Both must satisfy the security constraints introduced to achieve the secure dependency. This type of secure dependency is represented with security constraints on both sides (see Figure 4-3-c).

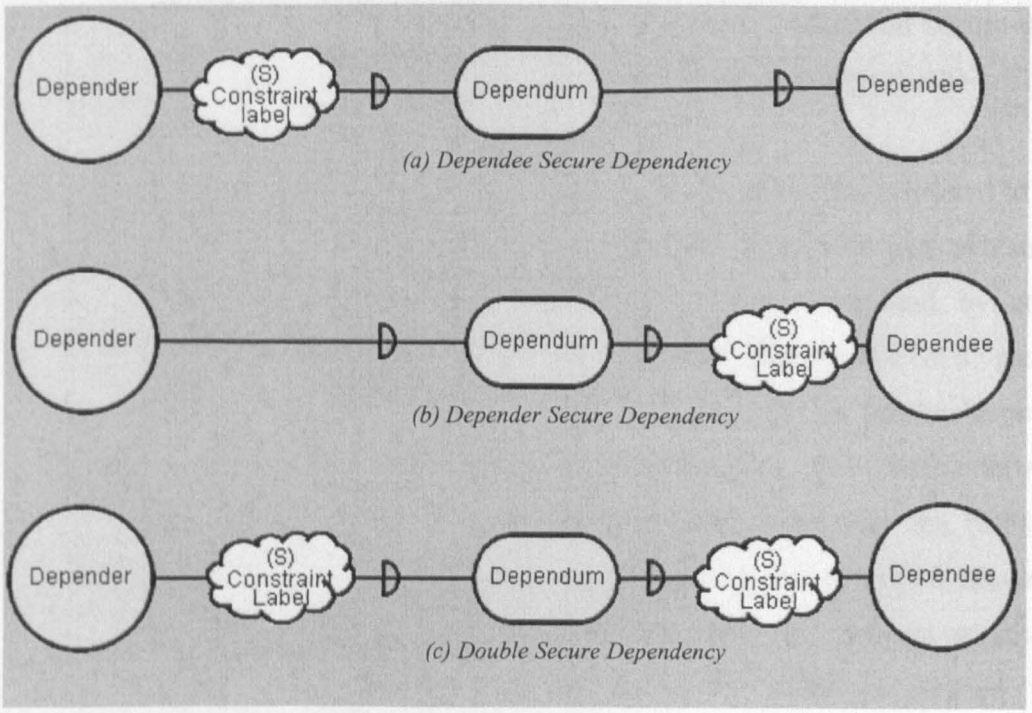


Figure 4-3: Graphical representation of secure dependencies

4.2.3 Secure entities

As mentioned above, the entities of the Tropos methodology need to be extended with security in mind. Therefore, in this research the term **secure entity** describes goals, tasks, and resources related to the security of the system. In other words, a secure entity represents a secure goal, a secure task or a secure resource.

A **secure goal** represents the strategic interests of an actor with respect to security. Secure goals are mainly introduced in order to achieve possible security constraints that are imposed to an actor or exist in the system. However, a secure goal does not particularly define how the security constraints can be achieved, since alternatives can be considered. As an example, consider an actor that is imposed a security constraint to provide information only if authorisation has been obtained. A secure goal (check authorisation) could be introduced to this actor to help towards the achievement of the imposed security constraint. However, this goal does not precisely define how the security constraint can be achieved. The actor could check the authorisation with many different ways.

The precise definition of how the secure goal can be achieved is given by a **secure task**. A secure task is defined as a task that represents a particular way for satisfying a secure goal. Consider, for instance, the above-introduced secure goal check authorisation. This goal can be satisfied by different security tasks such as check password or check digital signatures.

A **secure resource** can be defined as an informational entity that is related to the security of the multiagent system. Secure resources can be divided into two main categories. Those that display some security characteristics, imposed by other entities, such as security constraints, secure goals, secure tasks and secure dependencies. As an example, consider an actor who depends on another actor to receive some information (resource dependency). However, this dependency is restricted by the constraint only encrypted information. Therefore the resource involved in this dependency is considered secure since it is an encrypted resource. On the other hand, the second category of secure resources involves resources directly associated with the security of the system. For example, consider the authorisation details file of an agent of the system.

In addition, the graphical representation of the Tropos entities has been extended to enable it to model the secure entities. Secure entities are indicated by the presence of an S within brackets before the description of the entity as shown in Figure 4-4.

Figure 4-4: Graphical representation of secure entities

4.2.4 Secure capability

A **secure capability** represents the ability of an actor/agent to achieve a secure goal, carry out a secure task and/or deliver a secure resource. For example, consider an agent that is responsible for providing cryptographic services in a multiagent system. This agent should possess secure capabilities to decrypt incoming data and encrypt outgoing data. Another example is an actor responsible for providing authorisation services to an agency. Such an actor should be provided with secure capabilities to allow her to provide authorisation clearance or reject an authorisation request. A graphical representation of a secure capability is given in Figure 4-5. It must be noted that Tropos did not provide a graphical representation for the concept of capability. Therefore, this research introduced a graphical representation for capability and extended this representation, by following the same technique of introducing an S within brackets before the capability label, to depict secure capabilities as shown in Figure 4-5.

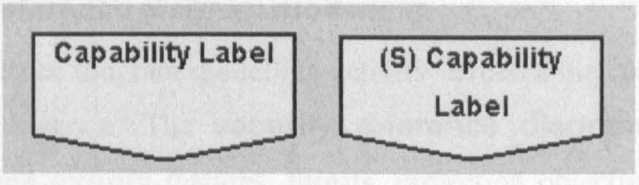


Figure 4-5: Graphical representation of a capability and a secure capability

4.3 MODELLING ACTIVITIES

The above-presented secure concepts form the basis of modelling security within the Tropos methodology. However, to make use of the above concepts different modelling activities contribute to the capturing and the analysis of the security

requirements of a multiagent system. Security-related modelling activities are divided into two main categories. Those newly introduced to the Tropos methodology, and those based on Tropos existing modelling activities that have been extended with respect to security modelling.

The first category includes the security reference diagram modelling, and the security constraints modelling, whereas the second category includes the secure entities modelling and the secure capability modelling.

The **security reference diagram modelling** involves the identification of security needs of the system-to-be, problems related to the security of the system, such as threats and vulnerabilities, and also possible solutions (usually these solutions are identified in terms of a security policy that the organisation might have) to the security problems.

The **security constraint modelling** involves the modelling of the security constraints imposed to the actors and the system, and it allows developers to perform an analysis by introducing relationships between the security constraints or a security constraint and its context.

The **secure entities modelling** involves the analysis of the secure entities of the system, and it is considered complementary to the security constraints modelling.

The **secure capability modelling** involves the identification of the secure capabilities of the actors and the agents of the system to guarantee the satisfaction of the security constraints.

These four modelling activities are presented in the following four sections.

4.3.1 Security reference diagram modelling

The security reference diagram modelling activity involves the construction of the security reference diagram. The **security reference diagram** represents the relationships between security features, threats, protection objectives, and security mechanisms. A security reference diagram is constructed after analysing the security requirements of the system-to-be and its environment and it is similar to the security catalogue first introduced by Yu and Cysneiros [Yu02]. The main difference lies in the concepts, such as security features, protection objectives and security mechanisms, introduced by the security reference diagram and also on the integration

of the security reference diagram within the development stages of the Tropos methodology.

The main purpose of the security reference diagram is to allow flexibility during the development stages of a multiagent system and also to save time and effort. Many systems under development are similar to systems already in existence. Therefore the security reference diagram can be used as a reference point that can be modified or extended according to specific needs of particular systems.

The analysis done during the construction of the security reference diagram can be used later in the development process to identify security constraints that must be introduced to the system-to-be (by taking into account the security needs of the system) and also by identifying possible means (security mechanisms) that contribute towards the satisfaction of the security constraints that are introduced to the system.

The notation of the security reference diagram can be adapted to reflect the notation of the methodology that the diagram is integrated. This is very useful since it allows developers to work with well-known concepts and allows them to use the same concepts throughout the development process. In this work, concepts from the Tropos methodology such as soft-goals, goals and tasks are used to model security features, protection objectives and security mechanisms respectively.

4.3.1.1 Nodes of the security reference diagram

For the construction process of the security reference diagram the developer considers the security features of the system-to-be, the protection objectives of the system, the security mechanisms, and also the threats to the system's security features.

Security features (also protection properties) represent features associated to security that the system-to-be must have. In this work the concept of a soft-goal is used to capture security features on the security reference diagram. This decision was taken because the concept of soft-goal is used, in the Tropos methodology, to model quality attributes for which there are no a priori, clear criteria for satisfaction but are judged by actors as being sufficiently met [Yu02]. In the same sense, security features are not subject to any clear criteria for satisfaction. Examples of security features are privacy, availability, and integrity.

Protection objectives represent a set of principles or rules that contribute towards the achievement of the security features. These principles identify possible solutions to the security problems and usually they can be found in the form of the security policy of the organisation. In this work, protection objectives are modelled using the concept of goal. This has been decided because in the Tropos methodology a goal defines desired states of the world. In the same sense, a protection objective represents desired security states that the system must have. Examples of protection objectives are authorisation, cryptography and accountability.

Security mechanisms represent standard security methods for helping towards the satisfaction of the protection objectives. Some of these methods are able to prevent security attacks, whereas others are able only to detect security breaches. In this project, the concept of a task is used to model security mechanisms. This decision took place because in Tropos a task represents a particular way of doing something, such as the satisfaction of a goal. In the same sense, a security mechanism represents a particular way of satisfying a protection objective. It must be noticed that furthered analysis of some security mechanisms is required to allow developers to identify possible security sub-mechanisms. A security sub-mechanism represents a specific way of achieving a security mechanism. For instance, authentication denotes a security mechanism for the fulfilment of a protection objective such as authorisation. However, authentication can be achieved by sub-mechanisms such as passwords, digital signatures and biometrics.

Threats represent circumstances that have the potential to cause loss; or problems that can put in danger the security features of the system. Since Tropos notation does not provide any related concept to model threats, a new notation has been introduced (see Figure 4-6). Examples of threats are social engineering, password sniffing and eavesdropping attacks.

A graphical representation of the above-mentioned concepts of the security reference diagram is depicted in Figure 4-6.

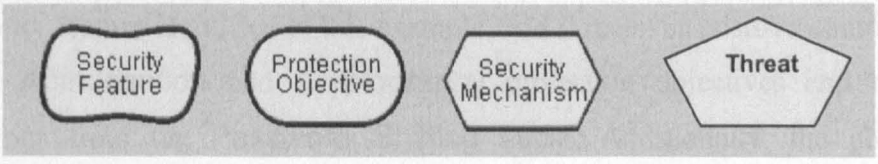


Figure 4-6: Graphical representation of nodes used in the security reference diagram

4.3.1.2 Links of the security reference diagram

The above-mentioned nodes of a security reference diagram are associated with the aid of two types of links (similar to the contribution links that can be found in the Tropos methodology): positive and negative contribution links. A positive contribution link associates two nodes when one node helps in the fulfilment of the other. Consider, for instance, a protection objective that contributes positively to the satisfaction of a security feature. A negative contribution link, on the other hand, indicates that a node contributes towards the denial of another node. As an example, consider the contribution of a threat to a security feature.

As a result, in every security reference diagram, each security feature identified receives positive contributions from different protection objectives and negative contributions from different threats.

Graphically a positive contribution link is modelled as an arrow, which points towards the node that is satisfied, with a plus (+) whereas a negative contribution link is represented as an arrow with a minus (-) as shown in Figure 4-7.

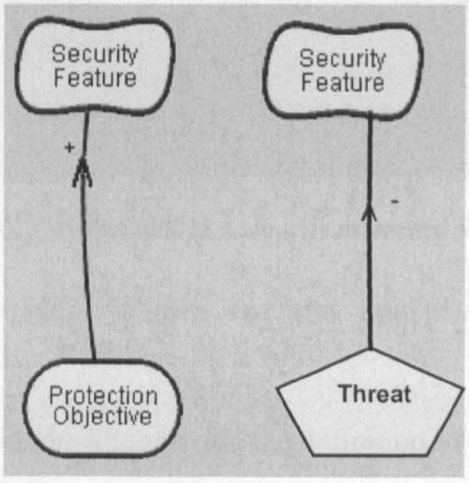


Figure 4-7: Positive and Negative Contribution links

4.3.1.3 An example of a security reference diagram

An example of a security reference diagram is given in Figure 4-8. Privacy is the only security feature identified in this example, and it receives positive contributions from the Authorisation and Cryptography protection objectives and negative contributions from the Password Sniffing threat. Additionally, the protection objectives are furthered analysed in terms of security mechanisms. Thus,

Cryptography can be achieved by different security mechanisms such as Encryption and Decryption. On the other hand, Authorisation can be achieved by Authentication. The Authentication security mechanism can be furthered analysed into sub-mechanisms such as Passwords, Digital Signatures and Biometrics.

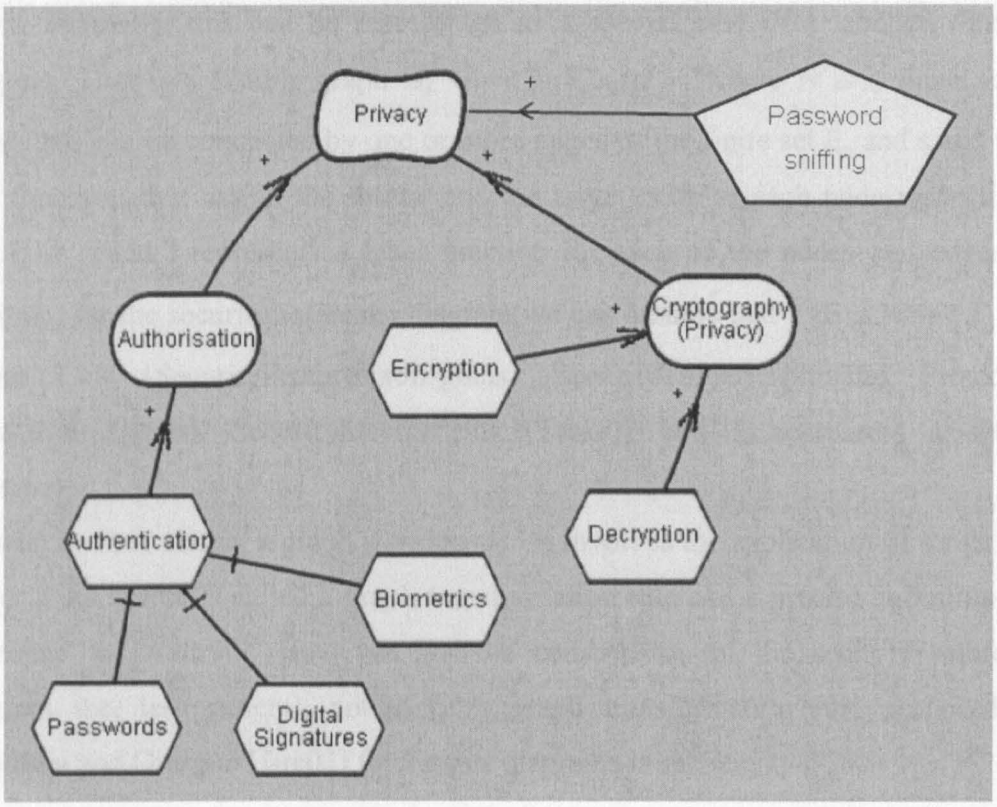


Figure 4-8: Example of a security reference diagram

4.3.1.4 A transformation system for the construction of the security reference diagram

The main aim of this section is to provide the definition of a transformation system for the construction of the security reference diagram in terms of a graph transformation system [Andr99]. Graph transformation allows the progressive derivation of the final diagram through subsequent more and more precise versions of it, according to the application of a set of rules to the diagram. Such an approach is very useful since it allows developers to precise inspect, by checking whether or not the diagram follows the construction rules, the development of the security reference diagram. The proposed transformation system is based on the graph transformation

system introduced by Andries et al. [Andr99], and the analysis proposed for Tropos' actor and goal diagrams by Bresciani and Giorgini [Bre02].

The security reference diagram can be seen as a graph that consists of a set of labelled nodes and a set of labelled directed edges, each of which connects a pair of nodes. Formally, this can be represented as a special case of a labelled directed diagram. That is a 5-tuple graph G, $G =< \mathrm{N}, \mathrm{E}, s, t, l >$, where N is a finite set of nodes that can be connected by one or more edges of the finite set E, and s and t are two functions that assign the source and the target node to each node respectively $s,t : \mathrm{E} \rightarrow \mathrm{N}$ and l represents a label function for each of the nodes and edges. In addition, for the security reference diagram we can assume that $l : \mathrm{E} \cup \mathrm{N} \rightarrow < T, L >$ where T = {SecurityFeatures(soft-goals), SecurityThreats (threats), Protection Objectives (goals), SecurityMechanisms (Tasks)} and L represents a set of identifiers.

As mentioned above, a graph transformation involves the application of a rule to a graph. Such a rule is called a graph transformation rule and a precise definition can be found in [Andr99]. However, for the construction of the security reference diagram the, less general, notion of a graph transformation rule proposed by Bresciani and Giorgini [Bre02] for Tropos diagrams is sufficient.

A graph transformation rule is a pair $r = \langle L, R \rangle$, where L and R are graphs called the left-hand-side (*LHS*) and the right-hand-side (*RHS*) of the rule. From the analysis done by Bresciani and Giorgini [Bre02] it derives that the application of rule r to a graph G results in a new graph H, $G \overset{r}{\Rightarrow} H$ according to the following three steps:

1. Chose an occurrence isomorphism from L onto a sub-graph G' of G, where G' is a sub-graph of a graph G if and only if $G' \cap G$ is well defined and $G' \cap G = G'$.
2. Delete from G the images of L with no counter-images in $L \cap R$, and obtain the context graph $D = G \setminus i(L \setminus R)$.
3. Add to D the images of the terms of R not already in D. This results in $H = D \cup i(R \setminus L)$.

Therefore, a graph H can be obtained from a graph G by the application of a set of transformation rules $P = \{r_1, ..., r_n\}$ as $G \overset{P}{\Rightarrow} H$ *or* $\overset{P}{\Rightarrow}$ in the case G is the empty graph.

However, the derivation process is non-deterministic due to the choice of a particular rule, at each step. Additionally, the chosen rule might be applicable to several occurrences of the graph's *LHS* [Andr99]. Therefore, to control this kind of non-determinism during the construction of the security reference diagram, priority rules have been assigned. These rules, in priority sequence, are presented below.

Rule 1: *Introduce the security features to the diagram*

$LHS :< \{\}, \{\}, \{\}, \{\}, \{\} >$

$RHS :< \{n_1\}, \{\}, \{\}, \{\}, \{n_1 \mapsto < SF, * >\} >$

The application of this rule results in the introduction of a new security feature (*SF*) in the *RHS* graph.

Rule 2: *Introduce the security threats and associate them with the security features*

$LHS :< \{n_1\}, \{\}, \{\}, \{\}, \{n_1 \mapsto < SF, * >\} >$

$RHS :< \{n_1, n_2\}, \{e_1\}, \{e_1 \mapsto n_2\}, \{e_1 \mapsto n_1\}, \{n_1 \mapsto < SF, * >, n_2 \mapsto < ST, * > e_1 \mapsto < NegCon, \varepsilon >\}$

The application of this rule results in the introduction of a security threat (ST) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 3: *Introduce the protection objectives and associate them with the security features*

$LHS :< \{n_1\}, \{\}, \{\}, \{\}, \{n_1 \mapsto < SF, * >\} >$

$RHS :< \{n_1, n_2\}, \{e_1\}, \{e_1 \mapsto n_2\}, \{e_1 \mapsto n_1\}, \{n_1 \mapsto < SF, * >, n_2 \mapsto < PO, * > e_1 \mapsto < PosCon, \varepsilon >\}$

The application of this rule results in the introduction of a protection objective (PO) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 4: *Introduce the security mechanisms and associate them with the protection objectives*

$LHS :< \{n_1\}, \{\}, \{\}, \{\}, \{n_1 \mapsto < PO, * >\} >$

$RHS :< \{n_1, n_2\}, \{e_1\}, \{e_1 \mapsto n_2\}, \{e_1 \mapsto n_1\}, \{n_1 \mapsto < PO, * >, n_2 \mapsto < SM, * > e_1 \mapsto < PosCon, \varepsilon >\}$

The application of this rule results in the introduction of a security mechanism (SM) in the RHS graph and the introduction of new edge(s) associated with this node.

Rule 5: *Decompose the security mechanisms to security sub-mechanisms*

$LHS :< \{n_1, n_2\}, \{\}, \{\}, \{\}, \{n_i \mapsto < SM, * >\} >$

$RHS :< \{n_1, n_2\}, \{e_1\}, \{e_1 \mapsto n_2\}, \{e_1 \mapsto n_1\}, \{n_i \mapsto < SM, * >, e_1 \mapsto < AND - DEC, \varepsilon >\}$

The application of this rule results in the introduction of new node(s) and edge(s) associated with the security mechanisms of the diagram.

4.3.1.5 Algorithm for the construction of the security reference diagram

Taking into account the above transformation system rules, the algorithm for the construction of the security reference diagram is given below.

```
BEGIN
Initialise Graph G (**should be empty in the initialisation process**)
  REPEAT
      REPEAT
            'choose rule 1';
            'choose an occurrence' i for the application of rule 1;
             G: = (G\i(L\R)+ i(R\L)
      UNTIL G = desired graph or no rule 1, for no occurrence i, remains;
      REPEAT
            'choose rule 2';
            'choose an occurrence' i for the application of rule 2;
            G: = (G \ i (L\R) + i (R\L)
      UNTIL G = desired Graph or no rule 2, for no occurrence i, remains;
      REPEAT
            'choose rule 3';
            'choose an occurrence' i for the application of rule 3;
            G: = (G \ i (L\R) + i (R\L)
      UNTIL G = desired Graph or no rule 3, for no occurrence i, remains;
      REPEAT
            'choose rule 4';
            'choose an occurrence' i for the application of rule 4;
            G: = (G \ i (L\R) + i (R\L)
```

UNTIL G = desired Graph or no rule 4, for no occurrence i, remains;

REPEAT

'choose rule 5';

'choose an occurrence' i for the application of rule 5;

G: = (G \ i (L\R) + i (R\L)

UNTIL *G = desired Graph or no rule 5, for no occurrence i, remains;*

UNTIL all rules are satisfied for all occurrences;

END

The general idea of the algorithm is to apply first all the security features, then the threats related to these features, then the protection objectives applicable to the security features, then the security mechanisms for the identified protection objectives and then the security sub-mechanisms.

Sometimes it might be the case that some extra nodes such as extra security features or extra threats are identified after the application of a particular rule. To avoid a delay in the analysis, it is convenient sometimes to allow some simple exceptions. Thus, it may be preferable to introduce the new node (by applying the corresponding rule) and then continue with the rest of the rules. For this reason, the outer **REPEAT** loop is necessary, since the application of one rule for a particular node, might require the application of a rule for another node.

As an example of how the proposed algorithm can be applied in the development of a security reference diagram, consider the security reference diagram of Figure 4-8. The application of the algorithm, for the construction of this security reference diagram is shown below, in which n_1 = privacy node, n_2 = password sniffing node, n_3 = authorisation node, n_4 = cryptography node, n_5 = authentication node, n_6 = encryption node, n_7 = decryption node, n_8 = passwords node, n_9 = digital signatures node, n_{10} = biometrics node.

BEGIN

OUTER REPEAT

Rule 1 Loop

$$\langle \{\ \}, \{\ \} \rangle \overset{R1}{\Rightarrow} \langle \{n_1\}, \{\ \} \rangle$$

Rule 2 Loop

$$\overset{R2}{\Rightarrow}\langle\{n_1,n_2\},\{n_2 \to n_1\}\rangle$$

Rule 3 Loop

$$\overset{R3}{\Rightarrow}\langle\{n_1,n_2,n_3\},\{n_2 \to n_1, n_3 \to n_1\}\rangle$$

$$\overset{R3}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1\}\rangle$$

Rule 4 Loop

$$\overset{R4}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3\}\rangle$$

$$\overset{R4}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5,n_6\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3, n_6 \to n_4\}\rangle$$

$$\overset{R4}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5,n_6,n_7\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3, n_6 \to n_4, n_7 \to n_4\}\rangle$$

Rule 5 Loop

$$\overset{R5}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3, n_6 \to n_4, n_7 \to n_4, n_8 \to n_5\}\rangle$$

$$\overset{R5}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8,n_9\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3, n_6 \to n_4, n_7 \to n_4, n_8 \to n_5, n_9 \to n_5\}\rangle$$

$$\overset{R5}{\Rightarrow}\langle\{n_1,n_2,n_3,n_4,n_5,n_6,n_7,n_8,n_9,n_{10}\},\{n_2 \to n_1, n_3 \to n_1, n_4 \to n_1, n_5 \to n_3, n_6 \to n_4, n_7 \to n_4, n_8 \to n_5, n_9 \to n_5, n_{10} \to n_5\}\rangle$$

END OF REPEAT

END

It is worth mentioning that the proposed security reference diagram transformation system is sound with respect to term graph rewriting in that for all term graphs G and H, $G \underset{u}{\Rightarrow} H$ implies $(G) \xrightarrow{n} term(H)$ where n is the number of paths from $root_G$ to each node u (proof given in [Plu02]).

4.3.2 Security constraint modelling

The **security constraint modelling** involves activities such as security constraint delegation and assignment, and also involves analysis that results in the identification of more detailed and precise security constraints and in the discovery

of secure goals that are introduced, to the actors, to help towards the satisfaction of security constraints.

More likely a developer will employ these activities in an iterative way, and will combine them with other modeling activities, such as goal, soft-goal, or tasks transformations, to allow the definition of the system-to-be according to the security constraints imposed. It depends on the designer to decide which activity must be employed at which stage of the system development. This is because the main aim of these processes is not to restrict the designer to a step-by-step development of the system-to-be, but rather to provide a framework that allows the developer to go from a very high level design to a more precise and defined version of the system.

4.3.2.1 Security constraint delegation and assignment

Security constraint delegation and assignment activities regard cases in which a security constraint is delegated from one actor to another (delegation) and when a security constraint is assigned to a specific goal of an actor (assignment).

When security constraints are imposed to a dependency, restrictions can be introduced to the actors that are part of this dependency. However, it can be the case that an actor delegates a security constraint imposed to them to another actor (through a dependency). This situation is known as security constraint delegation. As an example, consider a Patient that depends on their general practitioners to Receive Care as shown in Figure 4-9. A security constraint could be imposed to the General Practitioner to Keep Patient's Data Anonymous. However, the General Practitioner delegates the responsibility of providing care to a Nurse along with the security constraint Keep Patient's Data Anonymous.

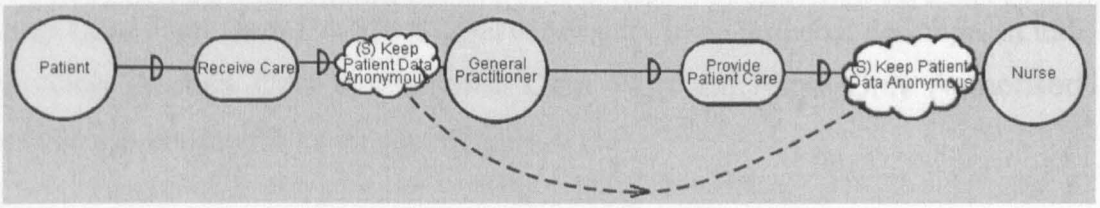


Figure 4-9: Example of a security constraint delegation

In case the security constraint is not delegated to another actor, further analysis is required to identify the goals of the actor that the security constraints restrict. This case is known as security constraint assignment. The assignment of a security

constraint to a goal is indicated with a contribution link that carries the "restricts" tag. Consider, for instance, the above example in which the Nurse has been imposed the security constraint to Keep Patient's Data Anonymous. Such security constraint could restrict some possible goals of the Nurse such as Share Patient Information. Therefore, the security constraint is assigned to this goal as shown in Figure 4-10.

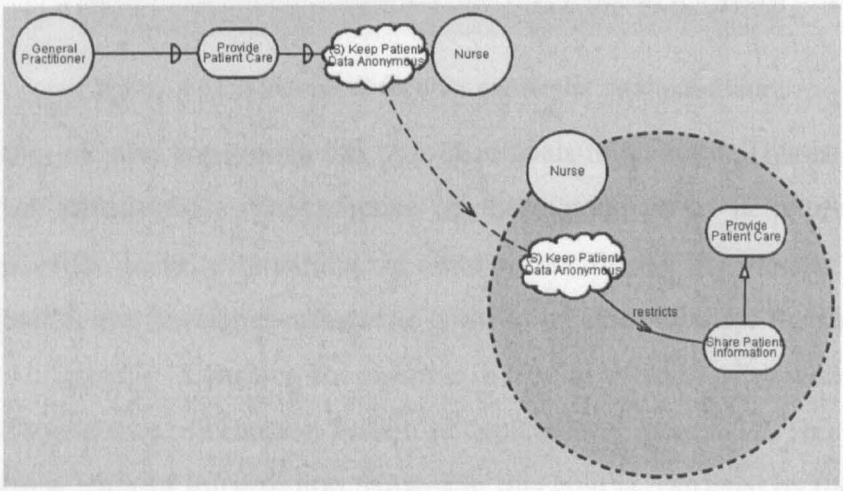


Figure 4-10: Example of a security constraint assignment

4.3.2.2 Security constraint analysis

When a security constraint is imposed to a goal (or task), two analysis processes are employed. Security constraint decomposition, which aims to further decompose the security constraint, and secure goal introduction, which identifies possible secure goals that the constraint might introduce to the system.

A security constraint can be decomposed to security sub-constraints, which define more precisely a security constraint. As an example, consider the security constraint Keep Care Plan Data Private. Such a constraint can be furthered decomposed into the Allow Access Only to Personal Care Plan and Allow Only Authorised Access sub-constraints as shown in Figure 4-11.

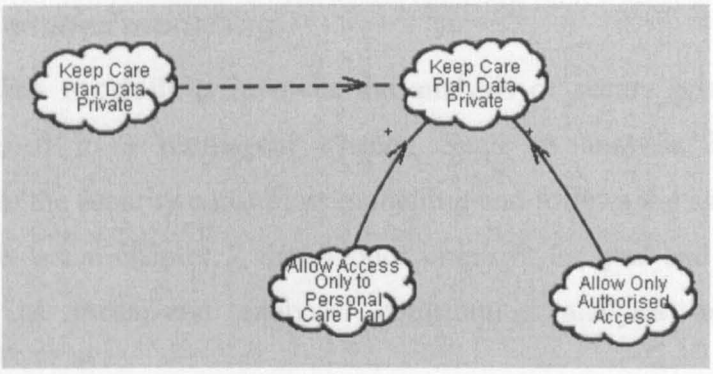


Figure 4-11: Example of security constraint decomposition

Furthermore, security constraints can introduce goals to an actor. This is known as secure goal introduction. The purpose of these goals is to help towards the achievement of the security constraint. In other words, during the process of secure goal introduction, the developer refines the goals of an actor to allow the satisfaction of a security constraint. Consider, for example, a Social Worker actor who is part of a health and social care information system as depicted in Figure 4-12. This actor has a goal to Share Patient Information. However, this goal is restricted by the security constraint Share Information Only If Consent Obtained. A secure goal, Obtain Patient Consent, is introduced to the actor to help towards the achievement of the security constraint (and therefore to help towards the achievement of the goal of the actor without endanger the security constraint). Since the secure goal helps towards the satisfaction of the security constraint, a positive contribution link is used.

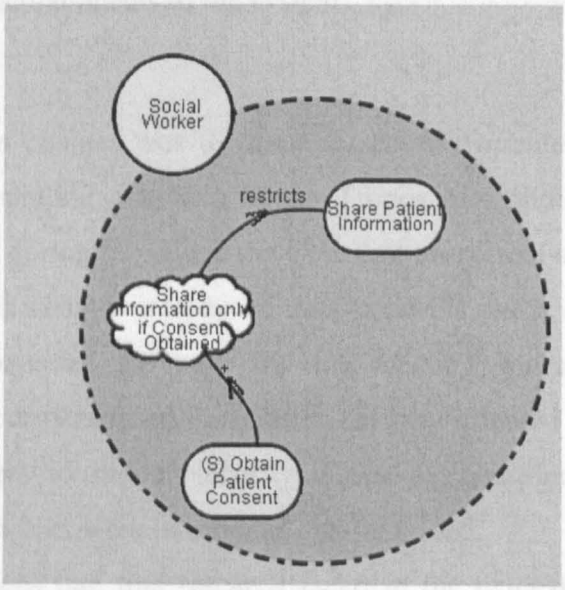


Figure 4-12: Example of secure goal introduction

4.3.3 Secure entities modelling

Secure entities modelling involves the analysis of secure goals, tasks and resources identified in a multiagent system. Such an analysis is considered complementary to the security constraints modelling and follows the same reasoning techniques, presented in chapter 3, that Tropos employs for goal and task analysis [Bre02a], such as means-end analysis, contribution analysis and AND/OR decomposition.

In particular, means-end analysis aims at identifying secure tasks and resources that provide means for achieving a secure goal. Contribution analysis permits developers to identify secure goals that contribute positively or negatively to the secure goal being analysed and AND/OR decomposition provides an AND/OR decomposition of a secure goal and/or task into sub-goals and sub-tasks respectively.

4.3.4 Secure capability modelling

The modelling of secure capabilities involves the identification of the secure capabilities of the multiagent system's actors to guarantee the satisfaction of the security constraints. **Secure capabilities modelling** takes place together with the capabilities modelling during the architectural design. Secure capabilities can be identified by considering dependencies that involve secure entities in the extended actor diagram. When identified, the secure capabilities are furthered specified in terms of plans of particular agents of the system.

4.4 SUMMARY

The purpose of this chapter was to describe security oriented extensions to the concepts and the modelling activities of the Tropos methodology to enable it to model security issues during the whole development process of a multiagent system. To fulfil this aim this chapter introduced new concepts, such as the concept of a constraint, and it extended the new and the existing concepts of the Tropos methodology with security in mind. In addition security-oriented modelling activities that enable developers to model security issues by considering the previously presented security concepts were introduced and described.

One of the challenges that this research faced in the extension, with respect to security, of the concepts and the modelling activities of the Tropos methodology was

the necessity to keep the modifications to the concepts and the modelling activities of the methodology to a minimum, in order to make the extensions easily understandable by developers familiar with the Tropos methodology and also to allow the usage of the same concepts and notations throughout the development process.

To meet this challenge, only the concept of a security constraint was newly introduced whereas the rest of the security concepts were extensions (redefinitions with security on mind) of Tropos existing concepts. On the other hand, extending the current notation by adding an S within brackets on the root concepts of the Tropos methodology to enable the modelling of the concepts related to security is a technique often used in this research. Such an approach introduces two important advantages. Firstly, it imposes minimum modifications in the notation of the methodology and therefore makes it easy to understand by developers familiar to the Tropos methodology and secondly, extending the notation like this allows further extensions. For instance some developers might find it useful to analyse constraints related to the performance of multiagent systems. Such constraints could be indicated by introducing, in the standard constraint notation, a P within brackets. This allows developers to differentiate the different categories of constraints and therefore analyse more precisely the multiagent system-to-be.

However, as mentioned in chapter 2, a security-oriented approach is required to guide developers in employing the presented concepts and modelling activities when developing multiagent systems. The following chapter illustrates such a process and it describes how it can be integrated within the Tropos methodology.

CHAPTER 5 A SECURITY-ORIENTED PROCESS

The previous chapter introduced concepts and modelling activities that enable developers to model security issues during the development of multiagent systems. However, a process is required to guide developers in employing the presented concepts and modelling activities when developing multiagent systems. The main aim of this chapter is to describe such a process.

The security-oriented process proposed by this research is mainly divided into four sub-activities; (1) The identification of security requirements of a multiagent system; (2) the selection amongst alternative architectural styles for the system-to-be according to the identified security requirements; (3) the development of a design that satisfies the security requirements of the system; (4) and the attack testing of the multiagent system under development. The first four sections, 5.1 to 5.4, of this chapter provide information about each of these activities.

Moreover, this chapter describes in section 5.5 how the consistency of the security-oriented process can be checked, and also it outlines in section 5.6 how the Tropos methodology stages can be refined to include the proposed security-oriented process. Section 5.7 summarises the chapter.

5.1 IDENTIFYING THE SECURITY REQUIREMENTS OF THE SYSTEM

The first step in the proposed security oriented process is to identify the security requirements of the system. Security requirements are identified by employing the modelling activities described in the previous section, such as security reference diagram construction, security constraints and secure entities modelling.

The process of identifying the security requirements of the system is basically one of analysing the security needs of the stakeholders and the system in terms of security constraints imposed to the system and the stakeholders, and identify secure goals and entities that guarantee the satisfaction of the security constraints.

The first step in the security process consists of the construction of the security reference diagram according to the principles and the techniques described in section 4.3.1. When the security reference diagram is complete, the analysis of the actors of the multiagent system takes place and security constraints are imposed to the actors of the system. In addition, security constraints are imposed to the system-to-be, with the aid of the security reference diagram.

When the security requirements of the system-to-be and the involved actors have been identified, the next step in the process consists of identifying an architectural style for the system that will satisfy the security requirements. The following section describes such a process.

5.2 SELECTING AMONGST ALTERNATIVE ARCHITECTURAL STYLES

As mentioned in section 2.3.2.2 an important requirement of a security-oriented approach is to allow developers to explore different architectural designs or in other words, to allow developers to reason about alternative design solutions according to the security requirements of a multiagent system.

For this reason, this research has developed an analysis technique to enable developers to select among alternative architectural styles[15] using as criteria the non-functional requirements of the multiagent system under development. The proposed technique is similar to the evaluation process for organisational styles proposed by Kolp et al. [Kol01]. The main difference is that Kolp's process is based on a qualitative reasoning, while the technique proposed by this research is based on an independent probabilistic model, which uses the measure of *satisfiability* proposed by Giorgini et al. [Gio02]. S*atisfiability* represents the probability that a non-functional requirement will be satisfied. Therefore, the analysis involves the identification of specific non-functional requirements and the evaluation of different architectural styles against these requirements.

The evaluation results in contribution relationships from the different architectural styles to the probability of satisfying the non-functional requirements of the system. To express the contribution of each style to the satisfiability of each non-functional

[15] To avoid confusion it must be noted that architectural styles differ from architectures in that " a style can be thought of as a set of constraints on an architecture" [Bas98].

requirement of the system, a weight is assigned. Weights take a value between 0 and 1. For example, 0.1 means the probability that the architectural style will satisfy the non-functional requirement is very low (the style is not suitable for satisfying the requirement). On the other hand, a weight of 0.9 means the probability that the architectural style will satisfy the non-functional requirement is very high (the style is suitable for satisfying the requirement).

The weights of the contribution links are assigned after reviewing different studies, evaluations, and comparisons involving the architectural styles under evaluation. When the contribution weights for each architectural style to the different non-functional requirements of the system have been assigned, the best-suited architectural style is decided. This decision involves the categorization of the non-functional requirements according to the importance to the system and the identification of the architectural style that best satisfies the most important non-functional requirement using a propagation algorithm, such as the one presented by Giorgini et al. [Gio02].

In case that two or more non-functional requirements are of the same importance, the presented technique can be integrated with other analysis techniques, such as the Software Architecture Analysis Method (SAAM) [Kaz94], to indicate which architectural style is best suited for the system-to-be.

Although the presented technique can be employed for the evaluation of architectural styles according to different non-functional requirements of a multiagent system, this research investigates the integration of security analysis within the development cycle of multiagent systems, and as a result security requirements are considered the most important, in this thesis, and the basis for the choice of the architectural style. Therefore, the technique has been focused on evaluating different architectural styles by considering security as the most important non-functional requirement of a multiagent system.

To demonstrate the above-presented technique, consider two architectural styles, a hierarchical style – *client/server* – and a mobile code style – *mobile agents*. In addition, for this example, consider that privacy is the most important security requirement of the multiagent system-to-be and the one that the architectural styles are evaluating against. As shown in Figure 5-1, in this example, the architectural

style that satisfies most the privacy requirements of the system is the *client/server* style because it contributes higher towards the privacy requirement than the mobile agents style. Consider, for example, the Information Flow property. This property is easier to be damaged by employing mobile agents (weight 0.4) since possible platforms that a mobile agent could visit might expose sensitive information from the agent. This is due to the fact that the mechanisms focused on the protection of mobile agents from a server cannot prevent malicious behaviour from occurring [Jan99].

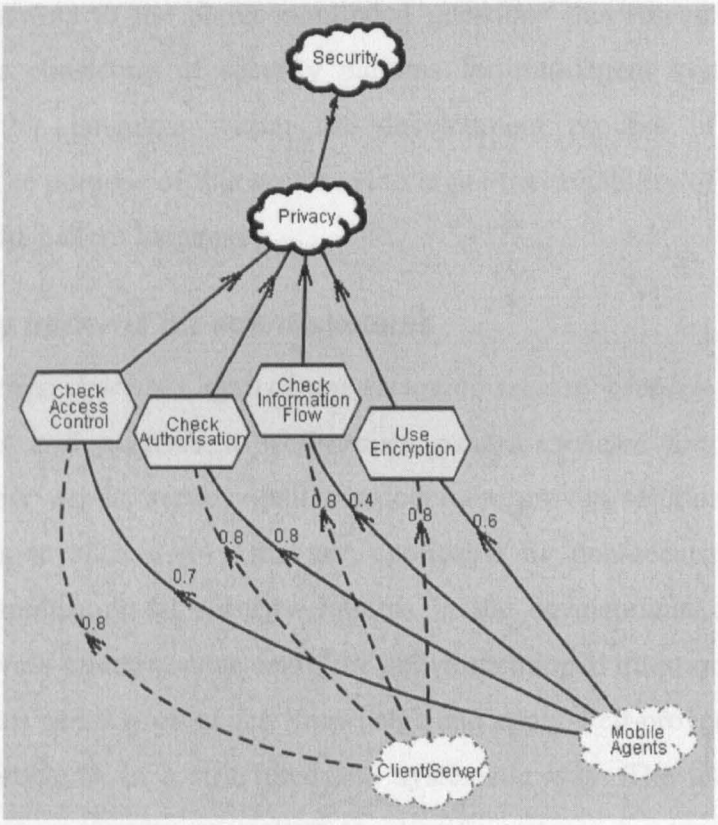


Figure 5-1: An example of selecting amongst architectural styles

On the other hand, in the case of the *client/server* style (weight 0.8) sensitive information is stored in the server and existing security measures could be taken to satisfy the Information Flow attribute.

5.3 TOWARDS A DESIGN THAT SATISFIES THE SECURITY REQUIREMENTS

As mentioned in section 2.3.2.1 one of the main reasons that security is not integrated within the development process of multiagent systems, is that developers who lack security expertise are involved in the development of multiagent systems. This situation gives rise to two critical questions. *How it can be assured that non-*

security specialists will have the knowledge to successfully transform security requirements to design? And *how the developer can be sure the proposed solution satisfies the security requirements of the system?* In projects that are stressed on time and budget developers must "acquire" security knowledge within a short timeframe and make sure that the system developed will work according to the requirements. A developer should know which designs are suitable for the problem, and any sequences an existing design will force to their system.

To provide answers to the above-mentioned questions this research proposes a pattern language consisting of security patterns for multiagent systems and the integration of this language within the development process of the Tropos methodology. The purpose of this section is to argue the suitability of the approach and to describe the pattern language.

5.3.1 Security patterns for agent systems

"*A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution*" [Schu01]. In other words, security patterns document proven solutions to security related problems in such a way that are applicable by non-security specialists. Therefore, the application of security patterns in the development of multiagent systems can provide effective answers to the above-mentioned questions, since non-security specialists can rely on expert knowledge and apply well-proven solutions to solve security problems in a structured and systematic way. The use of security patterns enables non-security specialists to identify patterns for transforming the security requirements of their system into design, and also be aware of the consequences that each of the applied security patterns introduce to their system. Additionally, because security patterns capture well-proven solutions, it is more likely that the application of security patterns will satisfy the security requirements of the system.

Nevertheless the advantages of security patterns have been mainly neglected during the development of multiagent systems [Mou03b]. One of the reasons is the lack of documented security patterns for the development of multiagent systems. As stated by Deugo [Deu01], documenting some techniques as patterns, does not mean to

document the problem and the solution, since such documentation can be found in many papers describing the techniques, but rather to provide a deeper understanding of the forces and the context of the problems that give rise to the proposed solutions.

As a result, the literature provides only references [Fern01, Fern02, Yod97] to object oriented security patterns. Although these patterns show similarities with possible agent oriented security patterns, the social nature of agent-based systems and the different security requirements due to unique characteristics in multiagent systems, such as autonomy, mobility, openness and trust, introduces a void that existing patterns have not filled [Mou03b].

Therefore, it is important to develop a pattern language consisting of security patterns for multiagent systems. The next section describes a pattern language consisting of security patterns for multiagent systems.

5.3.2 The pattern language

A *pattern language* is a set of closely related patterns that guides the developer through the process of designing a system. Using a pattern language, a design starts as a "fuzzy cloud" that represents the system to be realised. As patterns are applied, parts of the system come into focus, each pattern suggesting new patterns to be applied that refine the design, until no more patterns can be applied [Bec94]. The quality of a pattern language itself depends, among other things, on its *cohesion* (how closely the patterns are related), *coverage* (how many of the designs in its application domain it can generate), and *navigability* (how easy to use and understandable the links between patterns are).

Therefore, a good pattern language for the development of secure multiagent systems should contain security patterns that are based on agent-oriented concepts, described in section 2.2.2.1, such as intentionality, autonomy, sociality and identity. Each of the patterns of the language should be explicitly defined and also the relations between them must be precisely identified. Additionally, the structure of the patterns should be described not only in terms of the collaborations and the message exchange between the agents, but also in terms of the social dependencies and the intentional attributes, such as goals and tasks, of the agents involved in the pattern.

This allows for a complete understanding of the pattern's social and intentional dimensions, two factors very important in agent-based systems.

It is important to mention that the presented language consists of design patterns. The main difference between this kind of patterns and others, such as analysis [Fow97], and architectural [Bus96] patterns, is mainly the detail and the abstractions used to describe each pattern. For instance, an *analysis pattern* captures a conceptual model in an application domain in order to allow reuse across applications [Fow97], whereas an *architectural pattern* expresses a fundamental structural organization or schema for software systems, and it provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them [Bus96]. In contrast, a *design pattern* provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context [Bus96].

Having these factors in mind, the pattern language developed by this research contains four new agent design patterns (only patterns hereafter) and also describes the relationship of these patterns with other existing patterns. In particular the language contains the AGENCY GUARD[16] that provides a single, non-bypassable, point of access to an agency, the AGENT AUTHENTICATOR that provides authentication services to an agency, the SANDBOX that allows an agency to execute non-authorised agents in a secure manner, and the ACCESS CONTROLER that allows an agency to provide access to its resources according to its security policy.

Figure 5-2 describes the relationship of the patterns of the language as well as their relationship with existing patterns. The diagram is a slight variant of a Unified Modelling Language (UML) class diagram (the analogy to UML breaks down sooner or later. For example, the pattern name often echoes the solution and can be about dynamic actions, while a class name tends to be a "thing", not an action). Each box indicates a pattern, where a solid-line box indicates a security pattern that belongs to

[16] Capitalisation indicates reference to patterns in the language developed by this research

the language developed by this research and a dashed-line box indicates a related existing pattern.

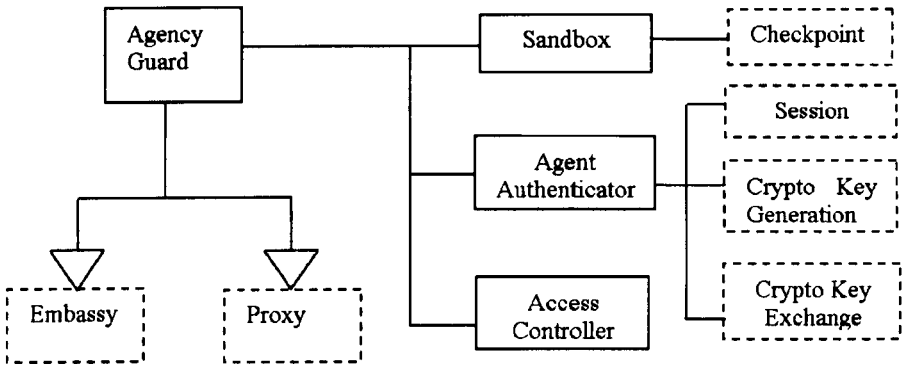


Figure 5-2: Relationships between the patterns of the language and other existing patterns

White triangles depict generalisations/ specialisation and solid lines associations of type uses/ requires. That way a hierarchy or a sequence of the security patterns is build, respectively. The AGENCY GUARD is the starting point of applying the patterns of the language and it is a variant of the *Embassy*[17] [Kol01] and the *Proxy* [Nor96] patterns. It uses the AGENT AUTHENTICATOR pattern to ensure the identity of the agents, the SANDBOX pattern in order to restrict the actions of agents, and the ACCESS CONTROLER pattern to restrict access to the system resources.

On the other hand, the SANDBOX pattern can implement the *Checkpoint* [Yod97] pattern, and the AGENT AUTHENTICATOR pattern can use the *Session* [Yod97] pattern to store credentials of the agent. Moreover, the AGENT AUTHENTICATOR employs the *Cryptographic Key Generation* [Leh02] and the *Cryptographic Key Exchange* [Leh02] patterns for further cryptographic actions.

For each of the patterns, the language provides the pattern name, the intent of the pattern, the context of the pattern, the description of the problem in which the pattern is applicable, the forces, the solution to the problem, the social dependencies, the consequences of applying the pattern, and any patterns related. These sections are mainly derived from sections proposed by Gamma et al. [Gam95], Buschmann et al. [Bus96], and Alexander [Ale79]. In particular, the name, intent, problems, solution,

[17] The use of italics in this section indicates patterns not developed by this research

and consequences sections are based on the definitions given by Gamma et al. [Gam95], the context and the forces sections are based on the definitions given by Alexander [Ale79], and the related patterns section is based on the definition given by Buschmann et al. [Bus96]. In addition to these, the proposed agent security design pattern template includes a social dependencies section that describes the social and intentional dimensions of the pattern.

The following section provides an analytical description of the four patterns of the language.

5.3.2.1 A description of the patterns

5.3.2.1.1 AGENCY GUARD (AG)

Intent: Provide a single, non-bypassable, point of access to the agency. The AGENCY GUARD defines a structure that makes unauthorized access to the agency difficult.

Context: A number of agencies exist in a network. Agents from different agencies must communicate or exchange information. This involves the movement of some agents from one agency to another or requests from agents belonging to an agency for resources belonging to another agency.

Problem: Many malicious agents will try to gain unauthorized access to agencies. If a malicious agent gains such an access, it can disclose, alter or destroy the data resided in the agency. Additionally, depending on the level of access the malicious agent gains, it might be able to completely shut off the agency or exhaust the agency's computational resources resulting in a denial of service to authorised agents of the agency. The problem becomes worse if many "back-doors" are available in an agency enabling malicious agents to attack the agency from many places. On the other hand, not all agents trying to gain access to the agency must be treated as malicious, but access should be granted based on the security policy of the agency.

Forces:

- The agencies provide access to subsequent resources. All of the corresponding resources have to be protected accordingly.
- More interfaces increase the flexibility and usability of an agent system, however, this also could result in duplicate code.

- A single interface can become complex when there are different types of authorization.

Solution: There must be a single point of access to the agency. When a Requester Agent wishes to access resources of an Agency or even move to this agency, its request is forwarded to the Agency Guard that is responsible to grant or deny the access requests according to the security policy of the agency. The Agency Guard is the only point of access in an Agency and it is always non-bypassable, meaning all the access requests are going through it.

Social Dependencies: A graphical representation involving the actors of the pattern and their social dependencies is shown in Figure 5-3. The Agency depends on the Agency Guard to grant/deny access to the agency. The Agency Guard grants / denies access according to the security policy. To obtain the security policy the Agency Guard depends on the Agency. The Requester Agent depends on the Agency Guard to obtain access to the Agency. For the Agency Guard to provide access to the Agency, a request must be sent from the Requester Agent.

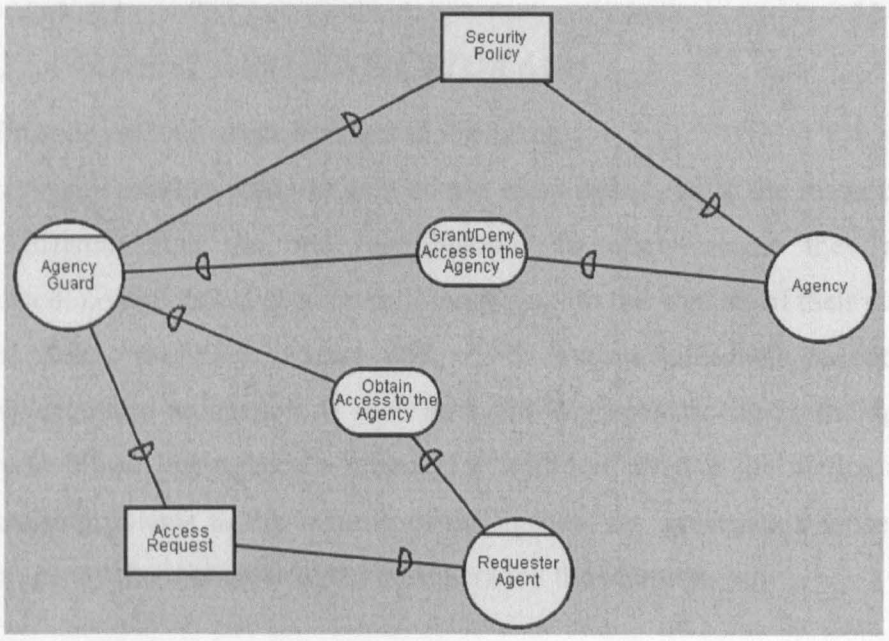


Figure 5-3: The AGENCY GUARD dependencies

Consequences:

+ Only the guard should be aware of the security policy of the agency, and it is the only entity that must be notified if the security policy changes (Not all the agents of the agency).

+ Only the guard must be tested for correct enforcement of the agency's security policy.
+ There are no many backdoors since there is only one point of access to the agency.
- Only one point of access to the agency can degrade performance of the agency.
- Only point of security, if it fails the security of the whole agency is in danger.

Related Patterns: The AGENCY GUARD has concepts of both the *Proxy* [Nor96] and the *Embassy* patterns [Kol01]. In addition, the AGENCY GUARD depends on the AGENT AUTHENTICATION pattern, in order to authenticate (verify the owner's identity) the agent requesting access. On the other hand, even if the agent is not authenticated the agency might decide to allow it to move to the agency but restrict its actions. For this reason the SANBOX pattern can be used. In traditional terms the concept of an AGENCY GUARD is related to the *Single Point of Access* [Yod97] and it is referred to as the *Reference Monitor* [Amo94, Fern02].

5.3.2.1.2 AGENT AUTHENTICATOR (AA)

Intent: Provide authentication services to the agency.

Context: Agents send requests to gain access to an agency or to the resources of an agency; different than the one they belong. To allow access they must be authenticated, i.e. they must provide information about the identity of their owners.

Problem: Many malicious agents will try to masquerade their identity when requesting access to an agency. If such an agent is granted access to the agency, it might try to breach the agency's security. In addition, even if the malicious agent fails to cause problems in the security of the agency, the agency will loose trust of the agent/agency the malicious agent masqueraded the identity.

Forces:

- Not all agents have to be authenticated or need all privileges.
- Both agencies and agents should be able to determine the identity of each other.
- Only weak authentication mechanisms, such as passwords, will not work in agent environments.

- Public authentication algorithms are widely tested and usually they are cryptanalysed. On the other hand, secret algorithms can (and usually will) be reverse-engineered.
- Cryptography is costly. More secure mechanisms usually lead to more expensive systems.

Solution: Agents have to be authenticated by the agency. By authenticating the agent; the Agency Guard makes sure it comes from an owner that is trustworthy for the agency. Each agent's owner and each agency have a public/private key pair. The Agent Authenticator can authenticate the agent on two cases: Firstly, when the agent is digitally signed with the owner's public key and secondly when the agent is digitally signed with the key of the agency that the agent resides. In the second case, the agent's agency would have authenticated the agent either if the owner signed the agent or if the agent was signed by the sending agency. In order for the second case to work, mutual trust must be involved between the sending and receiving agencies (each agency can be set up so it has a list of "trusted" agencies). In case that the Agent Authenticator does not trust the agency from which the agent comes from, it can reject the agent, or accept it with minimal privileges.

Social Dependencies: The graphical representation of the pattern dependencies is shown in Figure 5-4. The Requester Agent depends on the Agency Guard to obtain access to the agency. However, the Agency Guard cannot authenticate the Requester Agent by itself, so it depends on the Agent Authenticator to authenticate the agent. As a result, the Agent Authenticator receives a request for authentication from the Agency Guard when needed. In order for the Agent Authenticator to authenticate the Requester Agent, the Requester Agent should provide evidence of its digital signature. The Agent Authenticator has to send the notification to the Agency Guard when the agent is authenticated.

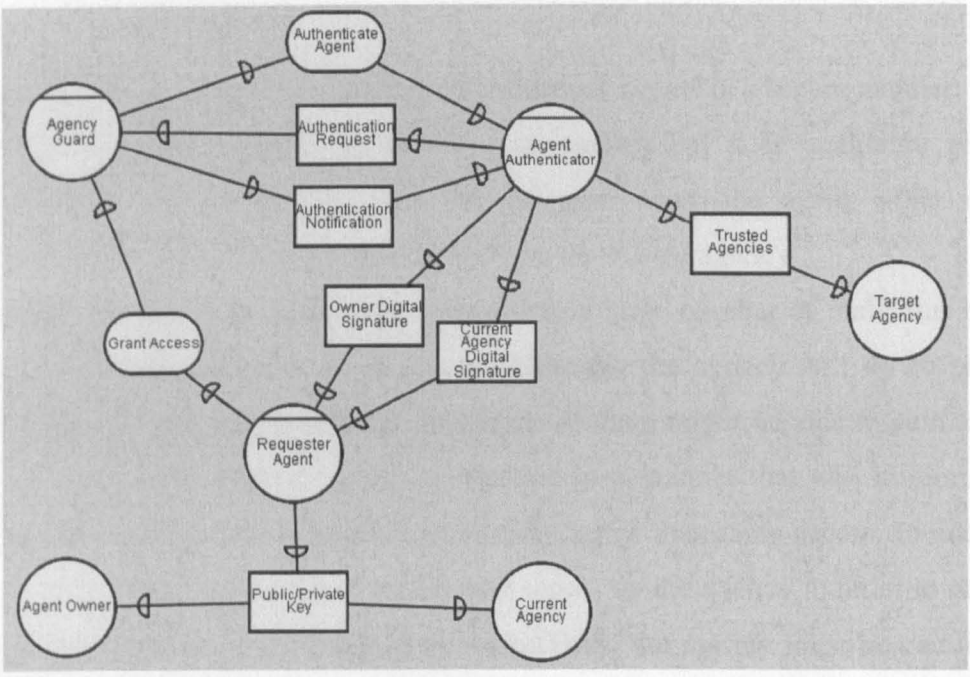


Figure 5-4: The AGENT AUTHENTICATOR dependencies

Consequences:

+ Authentication concerns are only dealt once. It is not necessary to make the agents of the system more complex by providing each one with an authentication mechanism.
+ Ensures that an agent is authenticated before actually request a resource from the agency.
+ During the implementation of the system, only the AGENT AUTHENTICATOR must be checked for assurance.
– A single point of failure. If the AGENT AUTHENTICATOR fails, the security of the whole agency is in danger.

Related Patterns: This pattern has some relations to patterns of the pattern language for cryptographic key generation [Leh02]. For example, a *Cryptographic Key Generation* is required. It is also important to have an appropriate *Cryptographic Key Exchange*. Furthermore, a *Session* can be used to store the credentials of an agent for subsequent requests [Yod97]. Moreover, the application of the SANDBOX pattern can be used to restrict the set of resources available to the agent.

5.3.2.1.3 SANDBOX

Intent: Allow the agency to execute non-authorised agents in a secure manner.

Context: An agent requests to move to an agency but it is unable to provide authentication certificates. This can be the case when the agent either is not authenticated or it has been authenticated by an un-trusted agency.

Problem: An agency is more likely exposed to a huge number of malicious agents that will try to gain unauthorised access. Although the agency will try to prevent access to those agents, it is possible that some of them might be able to gain access. Thus it is necessary for the agency to operate in a manner that will minimise the damage that can be caused by an unauthorised agent that gains access. In addition, some unauthorized agents might be allowed access by the agency in order to provide services the agency's agents cannot provide. Thus, the agency must be cautious to accept such unauthorised agents without put in danger its security.

Forces:

- An agent might need specific privileges to perform its task. However, it should not be allowed more rights than necessary.
- Not all agents are "security aware" and might act against the system's global policy.

Solution: Execute the agent in an isolated environment that has full control over the agent's ingoing and outgoing messages. Implementing such a sandboxing principle prevents any malicious agent from doing something is not authorised to do. The agent is allowed to destroy anything within the restricted environment but it cannot touch anything outside. The concept is similar to the Java programming language's use of a virtual machine environment and the chroot environment in UNIX. Malicious agents cannot do anything without first interacting with the operating system. Thus, SANBOX observes all system calls made by the agent and compare them to the agency-defined policy. If any violations occur, the agency can shut down the suspicious agent.

Social Dependencies: The graphical representation of the pattern dependencies is shown in Figure 5-5. The agency depends on the Sandbox agent for observing and controlling the agent's activities, and the Sandbox agent depends on the Agency to know adopted policies.

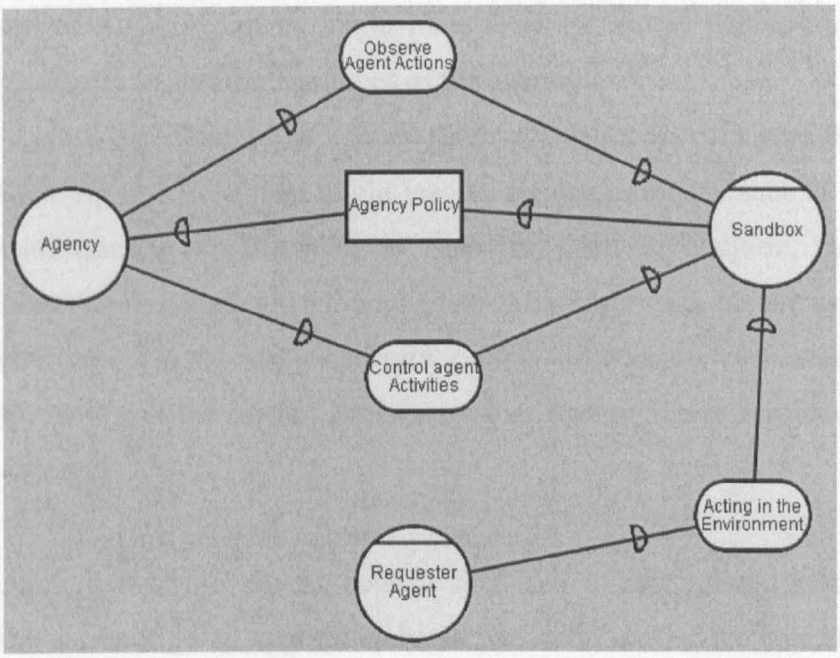


Figure 5-5: The SANDBOX dependencies

Consequences:

+ Agents not authorised but valuable for the agency can be executed without compromising the security of the agency.
+ Agency can identify possible attacks (by observing the actions of the agents in the SANDBOX).
– Some computational resources of the agency might be taken for non-useful actions (when non-useful agents are sandboxed).
– Introduce an extra layer of complexity on the agency.

Related Patterns: A checkpoint should be implemented within the SANDBOX in order to keep track of the exceptional actions and to decide what actions have to be taken based on the severity of the violation of the security policy (which defines what is allowed and what isn't). The SANBOX pattern is related to a similarly-named Java pattern [Jaw00].

5.3.2.1.4 ACCESS CONTROLER (AC)

Intent: Allow the agency to provide access to its resources according to its security policy.

Context: Many different agents exist in an agency. Those agents most likely will require access to some of the agency's resources in order to achieve their operational

goals. However, different agents might have different access permissions and are allowed access only to specific resources of the agency.

Problem: Agents belonging to an agency might try to access resources that are not allowed. Allowing this to happen might lead to serious problems such as disclosure of private information or alteration of sensitive data. In addition, more likely different security privileges will be applied to different agents on the agency. The agency should take into account its security policy and consider each access request individually. How can the agency make sure that agents access resources that are allowed to access?

Forces:

- It is unlikely that the access control facilities of all internal resources are activated and configured appropriately. In particular, out-of-the box installations offer standard services that can be misused by malicious agents. Even if there are access restrictions it is unlikely that they are consistent, especially when more than one administrator is involved and there are no "global" guidelines.
- Even worse, it could be assumed that most internal resources are not hardened. Experience shows that patches are not applied in time and that many, often unneeded services are running.
- Furthermore, it might happen that attacks cannot even be detected, as one cannot ensure that the audit facilities of the internal resources are activated and configured appropriately.

Solution: An Access Controler agent exists in the Agency. The Access Controler controls access to each resource. Thus, when an agent requests access to a resource, this request is forwarded to the Access Controler agent. The Access Controler checks the security policy and determines whether the access request should be approved or rejected. If the access request is approved the Access Controler forwards the request to the Resource Manager.

Social dependencies: The graphical representation of the pattern dependencies is shown in Figure 5-6. The Requester Agent depends on the Resource Manager for the resource, and the Agency depends on the Access Controler for checking the request. The Access Controler depends on the Agency for receiving the security

policies and for forwarding the request, which is forwarded to the Resource Manager in case it is approved.

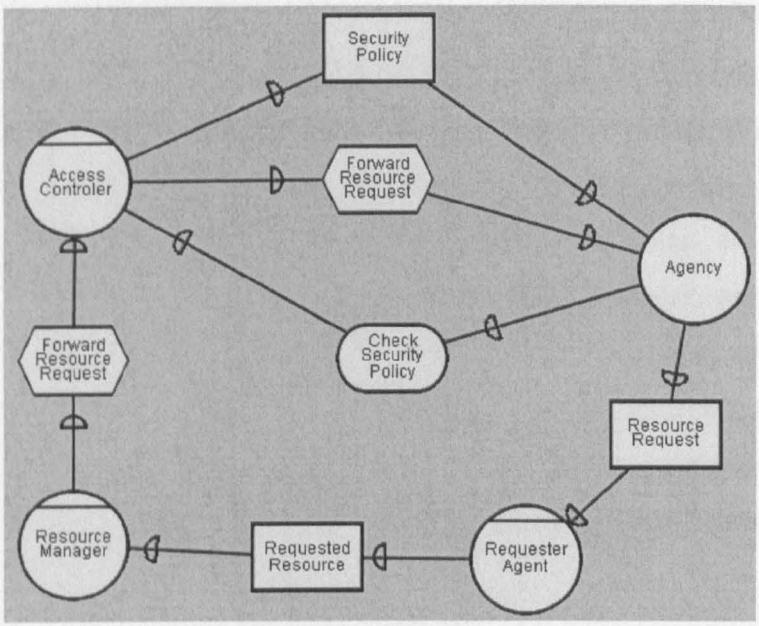


Figure 5-6: The ACCESS CONTROLLER dependencies

Consequences:

+ Agency's resources are used only by agents allowed to access them.
+ Different policies can be used for accessing different resources.
– One point of attack, if this fails the system access control system fails.

Related Patterns: The ACCESS CONTROLER pattern has been inspired by the *Role-Based Access Control* pattern presented by Fernandez [Fer01]. It is very similar (it can be thought of as a specialisation) to the AGENCY GUARD, but it focuses on access at resources within the agency rather than access to the agency.

5.3.2.2 An example of using the pattern language

As an example of employing the above presented pattern language in the development of a multiagent system, consider a system that must perform authentication and access control checks. In the case of the authentication checks, the multiagent system should be able to authenticate any agents that send a request to access information of the system, whereas in the case of the access control checks, the system should be able to control access to its resources.

To meet these goals, the AGENT AUTHENTICATOR pattern can be used to provide authentication checks and the ACCESS CONTROLER pattern can be used to perform access control checks. The AGENT AUTHENTICATOR satisfies the goal by authenticating each agent that tries to access the system, whereas the ACCESS CONTROLER is used to control access to the resources of the system. The use of these two patterns helps developers to delegate responsibilities of particular system security goals to particular actors defined by the patterns. Moreover, developers know the consequences that each pattern introduces to the system. In the presented example, for instance, the application of the AGENT AUTHENTICATOR pattern means that during implementation only the Agent Authenticator agent must be checked for assurance, whereas the application of the ACCESS CONTROLER means that different policies can be used for accessing different resources.

5.4 ATTACK TESTING OF THE MULTIAGENT SYSTEM UNDER DEVELOPMENT

The previous three sections of this chapter introduced a security-oriented process that allows the Tropos methodology to consider security issues during the development of multiagent systems. In particular, this process allows developers to identify the security requirements of a multiagent system, reason about a suitable architectural style, and successfully transform security requirements to design. However, an important issue is to test how the system under development copes with any possible attacks.

According to the IEEE Standard Glossary of Software Engineering [IEEE90], testability defines "*the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met*". Testing is widely considered an important activity that helps to identify errors in a system and techniques such as control and data flow testing, formal specifications, special testing languages, and test tools have been used for many years, in testing systems, and they are considered valuable solutions for many projects. However, most of these approaches are difficult to apply, they require

special training and skills, and they employ their own concepts and notations [Rys99].

Such requirements conflict with some of the requirements presented in section 2.3.2.2, according to which, a security-oriented approach should be clear and well guided, allow non-security specialists to consider security issues in the development process and it should employ the same concepts and notations throughout the development cycle of multiagent systems. Therefore, a technique, which is based on the use of scenarios and uses the same concepts and notations as the aforementioned in section 4.4 security-oriented process, has been developed and integrated within the security-oriented process to enable developers to test the system under development.

A scenario approach has been chosen since scenarios can be easily integrated within development methodologies and can be adapted to the methodology's notation and concepts. This is due to the fact that scenarios can be represented in various ways [Rys00]. In this research, scenarios are represented as enhanced Tropos diagrams.

Scenarios have increased in popularity among software engineers and have proven to be valuable for eliciting information about systems requirements, communicating with stakeholders and providing context for requirements [Rys00]. As a result, scenarios have been applied in many different areas of computer science research, such as software engineering [Pot94], business-process reengineering [Ant94], and user interface design [Car91]. In particular, many cases can be found in the literature [Rys99, Rys00, Lal95], where scenarios have been used for the validation of requirements.

In this research a scenario aims to test how the system copes with different kinds of security attacks. Therefore a scenario should include enough information about the system and its environment to allow validation of the security requirements. As such, a **Security Attack Scenario** (SAS) is defined *as an attack situation describing the agents of a multiagent system and their secure capabilities as well as possible attackers and their goals, and it identifies how the secure capabilities of the system prevent (if they prevent) the satisfaction of the attackers' goals.*

The presented approach aims to identify the goals and the intentions of possible attackers, identify through these a set of possible attacks to the system (test cases),

and apply these attacks to the system to see how it copes. By analysing the goals and the intentions of the attackers the developer obtains valuable information that helps to understand not only the *how* the attacker might attack the system, but also the *why* an attacker wants to attack the system. This leads to a better understanding on how possible attacks can be prevented. In addition, the application of a set of identified attacks to the system contributes towards the identification of attacks that the system might not be able to cope and this leads to the re-definition of the agents of the system and the addition of new secure capabilities to enable them to protect against those attacks.

The proposed scenarios-based analysis is similar to the work presented by Liu et al. [Liu02] that was discussed in the Introduction. However, there are many important differences. Liu's work is basically used to identify security requirements; the security attack scenarios in this work are used to test the security requirements of the system identified in the previous development stages. So a very similar idea is applied in a different stage of the development lifecycle. Liu argues that when the intentions of the attackers are identified the system can be equipped with countermeasures. However Liu does not mention how such countermeasures can be identified neither she provides a kind of process for applying these countermeasures to the system. Moreover, Liu's analysis takes place in a higher level than the one proposed by this research.

In this research, the secure capabilities of the actors of the system are known (and therefore a more precise idea of what security measurements the system has is given) and this allows the reasoning of possible security attacks according to those capabilities. In addition, in the presented approach test cases are considered. A process is provided that test each scenario for specific test cases, reason about the reaction of the system and take a final decision if the system can react to the specific attack. In cases that the system cannot react to the attack, possible countermeasures are discussed and extra secure capabilities are introduced to the actors of the system.

A security attack scenario involves possible attacks to a multiagent system, a possible attacker, the resources that are attacked, and the agents of the system related to the attack together with their secure capabilities. An attacker is depicted as an agent who aims to break the security of the system. The attacker intentions are

modelled as goals and tasks and their analysis follows the same reasoning techniques that the Tropos methodology employs for goal and task analysis. Attacks are depicted as dash-lined links, called attack links, which contain an "attacks" tag, starting from one of the attacker's goals and ending at the attacked resource.

For the purpose of a security attack scenario, a differentiation takes place between internal and external agents of the system. Internal agents represent the core agents of the system whereas external agents represent agents that interact with the system. Such a differentiation is essential since it allows developers to identify different attacks to resources of the system that are exchanged between external and internal agents of the system.

The process is divided into three main stages: creation of the scenario, validation of the scenario, and testing and redefinition of the system according to the scenario. Even though the presented process is introduced as a sequence of stages, in reality is highly iterative and stages can be interchanged according to the perception of the developers. The following three sub-sections describe each of these stages.

5.4.1 Scenario creation

There are two basic steps in the creation of a scenario. The first step involves the identification of the attackers' intentions and the possible attacks to the system and the second step involves identification of possible countermeasures of the system to the indicated attacks. The next two sections provide information about these steps.

5.4.1.1 Identify the intentions of possible attackers

During the first step, Tropos goal diagram notation is used for analysing the intentions of an attacker in terms of goals and tasks. Some of these goals can be identified by the threats modelled on the security reference diagram in Figure 4-8. For example, the threat Password Sniffing can introduce a goal Perform Password Sniffing to a potential attacker. However, other goals (apart from the ones introduced by the threats identified in the security reference diagram) could be derived from the analysis of a possible attacker's intentions. This is due to the fact that an attack is an exploitation of a system's vulnerability, whereas a threat is a circumstance that has the potential to cause loss or harm [Sch00]. Therefore, an

attack can lead to a threat only if the exploitation of the vulnerability leads to a threat. This means that some attacks can be successful but do not lead to threats as other system features protect the system. Figure 5-7 illustrates an example of the analysis of a possible attacker.

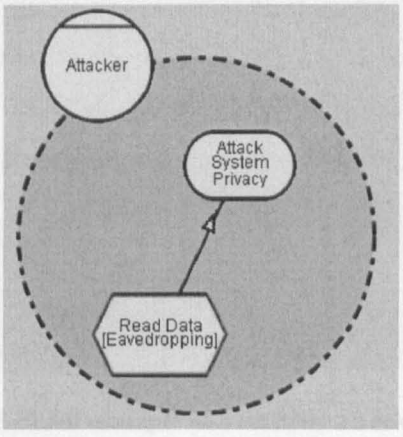


Figure 5-7: Example of a goal diagram analysing the intentions of an attacker

The main aim of the attacker presented is to attack the system privacy. Moreover, in this example the attacker employs a simple form of eavesdropping, by trying to read any information that is transmitted between the system and any external agents, to achieve their aim.

When the analysis of the attacker's intentions has been completed, possible attacks to the resources of the system are indicated using attack links.

5.4.1.2 Identify possible countermeasures

The next step in the creation of a security attack scenario involves the identification of the agents of the system that posses capabilities to prevent the identified, from the previous step, attacks. Therefore, the agents (internal and external) of the system related to the identified attack(s) are modelled. The secure capabilities, of each agent, that help to prevent the identified attacks are identified and dashed-links (with the tag "help") are provided indicating the capability and the attack they help to prevent. An example, of a security attack scenario is depicted in Figure 5-8. A System Internal Agent depends on the External Agent to obtain some Private Information. An Attacker aims to read the transmitted data (eavesdropping). However, the external and the internal agents have been assigned secure capabilities, such as encrypt and decrypt data, which helps towards the privacy of the data.

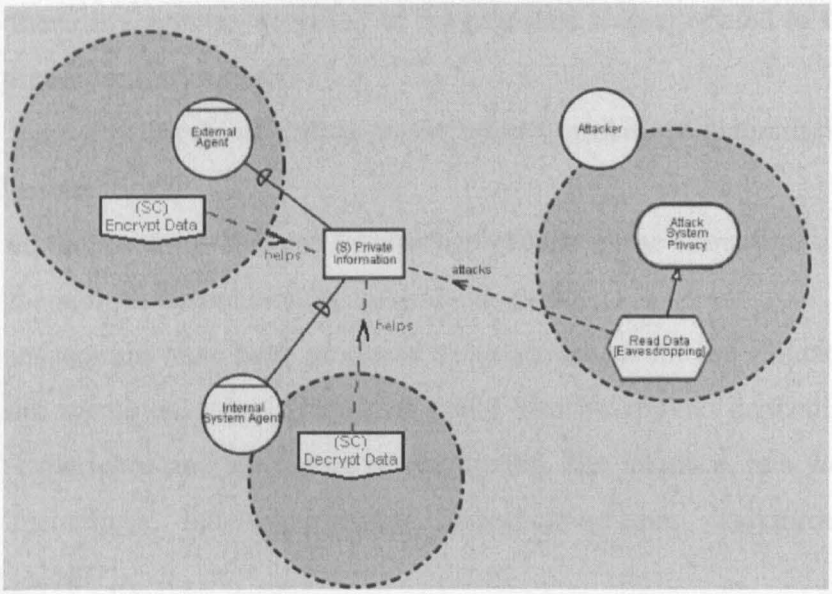


Figure 5-8: An example of a security attack scenario

5.4.2 Scenario validation

When the scenarios have been created, they must be validated. Therefore, the next stage of the process involves the validation of the scenario. Software inspections are proved as effective means for document-based validation [Kos97] and as such are the choice of this research for the validation of the security attack scenarios. The inspection of the scenarios involves the identification of any possible violations of the Tropos syntax and of any possible inconsistencies between the scenarios and the models of the previous stages. Such an inspection involves the use of validation checklists. Consider, for instance, the following checklist.

1. Is a name defined for each scenario?
2. Are agents represented using the correct notation?
3. Are attack links and help links correctly denoted?
4. Do the attack scenarios capture all possible attacks?
5. Do different scenarios exist for the same kind of attacks?
6. Are there any missing parts on the identified scenarios? (Any links missing or any agents missing?)
7. Are there any secure capabilities identified in the previous stages not present in the scenarios?

8. Are there any agents, identified in the previous stages, related to the attacks not present in the scenarios?
9. Are there any threats identified on the security reference diagram not present on the scenarios?
10. Are all the resources that can be attacked present in the scenarios?
11. Are the non-prevented attacks correctly marked?

Although inspections have been proposed by this research for the validation of the security attack scenarios, other techniques could also be applied depending on the developers' experience and the nature of the system. For instance, two well known validation techniques for requirements specification are walkthroughs and prototyping [Kos97].

5.4.3 Testing and redefinition of the system

When the scenarios have been validated, the next step aims to identify test cases and test, using those test cases, the security of the system against any potential attacks. Each test case is derived from a possible attack depicted in the security attack scenarios. Each test case includes a precondition (the state of the system before the attack), a system expected security reaction (how the system reacts in the attack), a discussion that forms the basis for the decision regarding the test case, and a test case result that indicates the outputs of the test case.

The test cases are applied and a decision is formed to whether the system can prevent the identified attacks or not. The decision whether an attack can be prevented (and in what degree) or not lies on the developer. However as an indication of the decision it must be taken into consideration that at least one secure capability must help an attack, in order for the developer to decide the attack can be prevented. Attacks that cannot be prevented are notated as solid attack links, as opposed to attacks that the system can prevent and which are notated as dashed attack links.

For each attack that it has been decided it cannot be prevented, extra capabilities must be assigned to the system to help towards the prevention of that attack. In general, the assignment of extra secure capabilities is not a unique process and depends on the perception of the developer regarding the attack dangers. However, a good approach could be to analyse the capabilities of the attacker used to perform the

attack and assign the system with capabilities that can revoke the attacker's capabilities.

5.5 CHECKING THE CONSISTENCY OF THE SECURITY PROCESS

The previous sections introduced a security-oriented process that helps developers identify security requirements, provide capabilities to the agents of the system to satisfy them, and test the system against possible attacks.

However, it is important, as described in chapter 2, to check that the process is consistent. Checking the consistency of the process is an important activity when developing software systems, multiagent or otherwise, and it is especially valuable when applied early in the development process, i.e. before implementation, as errors found during the analysis and design stages are much cheaper and easier to correct than errors found in later stages [Boe81]. The IEEE standard Glossary of Software Engineering Terminology [IEEE90] defines consistency as "*the degree of uniformity, standardisation and freedom, from contradiction among the documents or parts of a system or component*".

A number of different techniques [Boe84] are available to check consistency. These include manual techniques[18] such as manual cross-referencing, manual models, checklists and detailed scenarios and automated techniques such as automated cross-referencing, automated models and prototypes. According to an evaluation performed by Boehm [Boe84], manual cross-referencing constitutes an effective way to check the consistency. However, a set of consistency rules is required to allow cross-reference checking of a process.

Therefore, this research introduces a set of rules to help developers manually check the consistency of the security process.

[18] In this research, a manual approach has been chosen. To automate a process the manual process must be first defined. This is the aim of this project whereas automating the process and developing a tool is another project within the Tropos project initiative [Bre03].

5.5.1 Consistency rules

As mentioned by Nuseibeh et al. [Nus01] consistency rules can be identified by the definition of notations, the development methods, the development process model, local contingencies, and the application domain.

The rules, proposed by this research are expressed in a natural language and they can be applied more than once when checking the models and the process. This is due to the fact that the security-oriented process is iterative, and therefore the rules can be applied whenever iteration occurs.

Although, the presented set of rules provides a very good indication and substantially helps to check the consistency of the security models as well as the security process, it is not complete. As Nuseibeh et al. claim [Nus01], "*we do not expect to ever obtain a complete set of rules covering all possible consistency relationships in a large project. Rather, we regard the rule base as a repository for recording those rules that are known or discovered, so that they can be tracked appropriately*".

Consistency rules can be divided into inter-model rules, which help to check the consistency inside a model, and outer model rules, which help to check the consistency between the different models of a process.

It must be noted that this work considers only consistency rules that apply on the security related models and process and not rules for all the Tropos models and processes[19]. Therefore, this work provides consistency rules for all the security related modelling activities (inter-model rules) and for the whole security oriented process (outer-model rules).

The identified set of consistency rules helps developers to check the relationships between the components of the different security related models, such as the relationship between the security features and the threats in the security reference diagram, the consistency between same components appeared in more than one models, such as a security constraint that appears in the actors' model as well as in

[19] Readers interested in such rules please refer to [Per01, Giu02, Bre02b].

the goal model, and the consistency when delegation of components between actors takes place. Table 5-1 provides as an example[20] three consistency rules.

Table 5-1: Example of consistency rules

Rule Category	*Rule*
Security reference diagram rule	*Each protection objective and each threat that appear on the diagram must be associated with at least one security feature of the graph.*
General process rule	*Any security components that appear throughout the diagrams must have consistent names across the diagrams.*
Security constraint rule	*A security constraint modelled in the actors' diagram should appear in the appropriate actor's goal diagram.*

5.6 Refining the Tropos stages to include the security-oriented process

The previous sections introduced a security-oriented approach for the development of multiagent systems. However, to successfully complete the aims of this research, this process must be integrated within the development stages of the Tropos methodology.

For this reason, the Tropos development stages have been refined to accommodate the proposed security extensions. This section aims to discuss the integration of the security-oriented approach into the Tropos methodology stages.

- **Early requirements analysis stage**: During the early requirements analysis stage the security reference diagram is constructed and security constraints are imposed to the stakeholders of the system (by other stakeholders). In the actor's diagram, imposed security constraints are expressed in high-level statements. In the goal diagram the security constraints are furthered analysed as described in section 4.3 and secure goals and entities are introduced to the corresponding actors to satisfy them.

[20] Readers interested in the complete list of the rules please check Appendix A.

- **Late requirements analysis stage**: During the late requirements analysis stage, security constraints are imposed to the system-to-be (by reference to the security reference diagram). These constraints are further analysed according to the analysis techniques presented in section 4.3 and security goals and entities necessary for the system to guarantee the security constraints are identified.
- **Architectural design stage**: During the architectural design any possible security constraints and secure entities that new actors might introduce are analysed. Additionally, the architectural style of the multiagent system is defined with respect to the system's security requirements and the requirements are transformed into a design with the aid of security patterns. Furthermore, the agents of the system are identified along with their secure capabilities and security attack scenarios are used to test the security of the system under development.
- **Detailed design stage**: During the detailed design stage, the components identified in the previous development stages are designed with the aid of Agent Unified Modeling Language (AUML). In particular, agent capabilities and interactions taking into account the security aspects are specified with the aid of AUML. The important consideration, from the security point of view, at this stage is to specify the components by taking into account their secure capabilities. This is possible by adopting AUML notation.

5.7 SUMMARY

The purpose of this chapter was to introduce a security oriented process in the development of multiagent systems and integrate such a process within the development stages of the Tropos methodology, to enable it to model security issues during the whole development process of a multiagent system. To fulfil this aim this chapter described a security-oriented approach comprising of four main sub-procedures; (1) the identification of the multiagent system's security requirements; (2) the selection amongst alternative architectural styles; (3) the development of a design to satisfy the security requirements; (4) and the attack testing of the multiagent system under development.

In addition the chapter described a set of rules that enables developers to check the consistency of the security process, and it described how the proposed security extensions are integrated within the development stages of the Tropos methodology.

However, the proposed security approach can never be accepted if it cannot prove its validity in practise in a real-life case study. Employing the approach in a real life case study will enable to evaluate its successfulness, and the advantages it provides towards the development of secure multiagent systems.

Therefore, the following chapter illustrates how the proposed security extensions are applied in the development of the electronic single assessment process (eSAP) system, a real-life case study and also it provides a critical discussion/evaluation regarding the proposed security extensions.

CHAPTER 6 APPLYING THE EXTENSIONS: THE eSAP CASE STUDY

The previous two chapters introduced extensions to enable the Tropos methodology to model security issues during the development of multiagent systems. However to evaluate the proposed security-oriented approach and better understand its advantages, the approach must be applied to a real-life case study.

The aim of this chapter is to employ the proposed security-oriented approach in the development of the electronic single assessment (*eSAP*) system, a real-life case study that provided the initial motivation of this research. Section 6.1 describes the single assessment process and it outlines the motivations behind the development of the electronic single assessment process. A typical scenario regarding the single assessment process, which forms the basis for the development of the electronic single assessment process system, is presented in Section 6.2 and section 6.3 describes how the proposed security-oriented approach can be applied in the development of the electronic single assessment process system. Section 6.4 provides a critical discussion/evaluation regarding the proposed security-oriented approach and section 6.5 summarises the chapter.

6.1 THE SINGLE ASSESSMENT PROCESS AND THE MOTIVATION BEHIND THE ELECTRONIC SINGLE ASSESSMENT PROCESS

The assessment of health and social care needs is at the heart of good practice in the care of older people. Older people often have multiple impairments and health problems, and complex support systems involving several health and social care practitioners and family carers. Sharing of assessment information is important to avoid unnecessary repetition and to ensure that all relevant information is available to support effective care planning. Recognition of the need to share assessment information has stimulated standardisation of assessment methods. These in turn have been used to help standardise care planning and referrals following assessment.

In March 2001, the (English) Department of Health published its **National Service Framework** (NSF) for Older People's Services [Doh03]. The NSF sets national standards for the health and social care of older people, with an implementation plan, to be completed by 2005.

Standard 2 of the National Service Framework, which refers to person-centred care, includes requirements to establish a single assessment process for integrating the assessment of health and social care needs of older people. Local health and social care communities have to introduce standardised shared systems for assessing needs, with convergence towards a fully integrated and electronically based national system. The Department issued further guidance in February 2002, listing requirements for contact, overview, specialist and comprehensive assessment, and a range of assessment instruments, which could be used for these types of assessment.

Contact and overview assessments would typically be undertaken by front-line primary health and social care practitioners, with specialist and comprehensive assessments undertaken by secondary care specialists or multi-disciplinary teams. Contact and overview assessments would provide the basis for specialist and comprehensive assessment, with the breadth and depth of all assessments undertaken according to the perceived needs of the older person. It should be noted that elements of self-assessment are to be encouraged, and there is a strong emphasis on including the older person's views in establishing the focus of attention in assessing need and planning care.

Information technology has the potential to improve efficiency and effectiveness in the collection and sharing of assessment information. An information system, called hereafter the electronic single assessment process (***eSAP***) system, to support integrated assessment of the health and social care needs of the older person, should therefore build on contact and overview assessment in primary care, with maximum involvement of the older person in prioritising the assessment domains and in care planning.

6.2 A TYPICAL SCENARIO

Modelling the whole setting surrounding the single assessment process remains a major challenge not only for this research project, but also for everyone involved in health and social care. It is not only the large context that such a setting involves,

ranging from hospitals to police stations and jails, but also the variety of models and procedures that health and social care professionals employ in performing their duties.

On the other hand, it is widely known that when developing an electronic system, its boundaries should be precisely defined. Therefore, it was decided that in this research the development of the electronic single assessment process system should not be based on the whole setting but rather on a real-life scenario of the single assessment process. Such a development would identify the major actors of the system, and it will provide an analysis and design that would be the basis for a successful modelling of the whole setting. The following scenario has been used in this research for the analysis and design of the electronic single assessment process.

"An 81 years old lady, widow, lives in her house. Her daughter lives nearby but she has children of her own and therefore she is unable to provide full care to her mother. However, she sees her mother everyday.

The daughter visits the mother's General Practitioner (GP) to describe her concern about her mother's health. Her mother has become unsteady on her feet and may have had a number of falls. Single assessment process has been introduced, so the GP asks the daughter to complete the EasyCare [Phi97] *contact assessment and the information is entered into the GP's computer. The GP sees the daughter concerned about her mother's health and asks his practice nurse to visit the old lady to perform an overview assessment.*

The old lady's information is transferred to the nurse's computer along with referrals and instructions, e.g. the daughter of the patient is concerned about her mother's health so please perform an overview assessment. The nurse receives the information and arranges to visit the old lady by generating and sending a letter to the old lady and her daughter giving details about visit and ask availability. The daughter replies (also provides her mother's response) that the date/time is suitable.

The nurse visits the old lady and completes most of the EasyCare assessment except from the health promotion module. From the evaluation of the other modules the nurse concludes the old lady has a number of problems with her house, which increase the risk of falls, she needs help with dressing and also she is not getting the appropriate financial benefits. Then the nurse asks the old lady if the information

can be shared, and the old lady accepts. The nurse then produces a care plan summarising all the problems identified and the actions to be taken. She also makes two referrals one to a Social Worker (SW) - to check for a care assistant to help the old lady with dressing and to check about financial benefits- and a second to an Occupational Therapist (OT) -to perform a house assessment for need and adaptation. She then forwards the care plan and a summary of the problems to the General Practitioner and the care plan and contact information to the Social Worker and the Occupational Therapist. In addition, a copy is produced for both the old lady and her daughter and the care plan is signed.

Later, the old lady is visited by the Occupational Therapist who performs the house assessment and decides that the house needs to be adapted to the old lady's needs. The O.T. then makes a referral to the Equipment Services for equipment and also provides the contact information of the old lady. In addition, the O.T. forwards to the GP, nurse and the S.W. a copy of the house problems, needed equipment and informs them that a referral has been made to the Equipment Services.

The Social Worker visits the old lady and identifies that the old lady must apply for financial benefits. A form is produced, filled in, and sent to the Benefits Agency together with old lady's contact and bank information. In addition, the Social Worker agrees to employ a Care Assistant (C.A.) to help the old lady with dressing. A Care Assistant is identified and the Social Worker asks the old lady if she feels comfortable with the identified Care Assistant and the old lady agrees. Contact and overview assessment information is sent to the Care Assistant by the Social Worker. Also, because of the employment of a Care Assistant, the benefits must be adjusted. The social worker informs the benefits agency about it.

While the Care Assistant visits the old lady, she realises that the health promotion module of the overview assessment is not completed. She notifies the nurse and prompts the old lady to fill in the module. When the module is complete, the C.A. sends the information to the General Practitioner, the nurse and the Social Worker.

One of the observations of the Care Assistant was that the old lady didn't have her blood pressure taken for the last 5 years, so she alerts the nurse and the G.P. The General Practitioner receives the alert and makes a referral to the nurse to go and check the blood pressure of the old lady. The nurse visits the old lady to review all

the actions of the care plan and also check the old lady's blood pressure. The care plan is updated and for the time being the old lady gets everything she needs."

6.3 DEVELOPING THE *eSAP*

The above scenario provides the basis for the development of the system. As mentioned in previous chapters the first phase of the Tropos methodology is the early requirements analysis. It must be noticed that the presented development process is focused on the security-oriented extensions described in the previous chapters.

6.3.1 Early requirements analysis

During the early requirements analysis, the security reference diagram is constructed as described in section 4.3.1. For the construction of the diagram, the security features of the system must be identified together with protection objectives, security mechanisms and threats.

Security is a very important factor in the development of the electronic single assessment process, since security of personal health information is considered a priority by many health care unions in different countries of the world including England. This is due to the fact that in cases where patients (in the case of the ***eSAP*** older people) do not trust the security of the system, they will refuse to provide complete information about their health and social care needs, and this could lead to many problems such as wrong assessment of needs, which could lead to wrong care plans.

The advances on information technology and the introduction of nationwide networks have caused concerns about security to the health and social care professionals and the patients. The electronic single assessment process lies in this category, as it is intended to be used nationwide in England. Health and social care professionals and older persons are worried that using such a system introduces risks for the privacy (it is privacy that empowers the patient, rather than confidentiality that empowers the organisation. This distinction, although it is familiar to medical ethicists, is less familiar to the computer security world [And01]) of personal health and social care information. Therefore privacy of health and social care information, such as the health and social care plans used in the electronic single assessment process, is the number one security concern in such a system. According to the Good

Medical Practice, patients have a right to expect that you will not pass on any personal information, which you learn in the course of your professional duties unless they agree. In addition to that, the English government and health and social care unions have agreed that electronic health care records should be as well protected as the paper ones.

Other important concerns are integrity and availability. Integrity assures that information is not corrupted and availability ensures the information is always available to authorised health and social care professionals. If assessment information is corrupted or it is not available the care provided to the older people (in the case of the ***eSAP***) by the health and social care professionals will not be efficient or accurate. Therefore, it is necessary to find ways to help towards the privacy, the integrity and the availability of personal health and social care information.

From the above discussion it is derived that the main security features for the electronic single assessment process system are privacy, integrity and availability as shown in the security reference diagram in Figure 6-1.

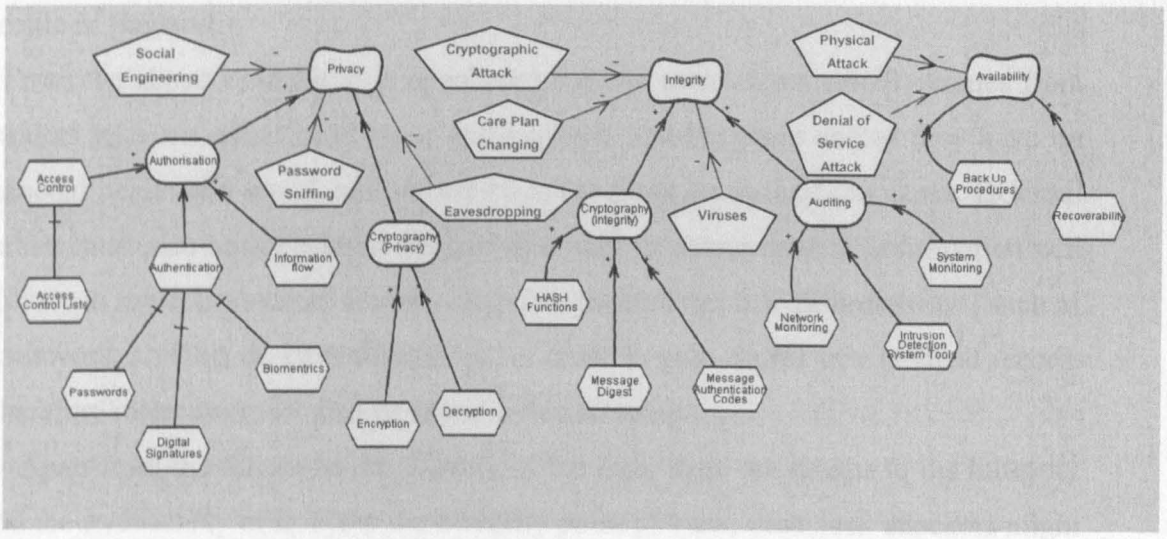


Figure 6-1: Security reference diagram

On the other hand, security threats to the electronic single assessment process (***eSAP***) are mainly the same as in any other medical system. According to Anderson [And01] the main threat to medical privacy is social engineering [Gra02]. According to this, a typical attack on a health and social care information system involves a private detective (or someone interested in obtaining personal health information) that calls in the health professional's office, introduces himself as a

doctor in an emergency or acute hospital and asks information about the medical record of a particular patient [And01]. One such case that had a big impact in England was the case that private health records were sold from as little as £150 [Eco99]. Private agencies were able to reveal complete medical files within three hours. The only information they required was the name, address, and the date of birth of the patient they were investigating. It was thought they were obtaining the records by the method of social engineering.

Furthermore, the size of the electronic single assessment process system and the large number of health and social care professionals that might be involved introduces the problem of data aggregation and increases the risk of social engineering or unauthorised access. The risk factor of private data to be accessed by unauthorised personnel increases by the number of people that have access to it and aggregating information increases this risk factor. It is easier to secure the data in a hospital that has records for 10,000 patients than secure a system, such as the electronic single assessment process, that will contain data of almost all the older people in England.

From the above example it is concluded that the main threat usually comes from insiders who are either careless or manipulated, and the more access they have on personal health and social care information, the more harm they can cause. External threats must also be considered. People who want to obtain medical information will also try to break the system security. Capable hackers can use different ways, such as password sniffing or eavesdropping, in order to gain access to a medical record. Therefore measures must also be taken in this direction.

Apart from the threats to the privacy of the data, there are threats to the integrity and the availability of it. From the integrity point of view, malicious attackers might change the content of medical care plans. In addition, cryptographic attacks can be used to manipulate messages sent between actors of a system or viruses can be created in order to affect the integrity of the information. From the availability point of view, physical attacks to the system are a main threat. An attacker tries to make the system unavailable by physically destroying a part of it. Moreover, denial of service attacks form a popular threat to the availability of the system. According to this, a number of compromised systems attack a single target. This initially results in

denial of service to the users of the targeted system, and later in the shut down of the system, therefore making the system unavailable.

When the security reference diagram for the system has been developed, the main actors should be identified. From the scenario presented in 6.2, the following actors are derived.

Older Person: The Older Person actor represents patients aged 65 or above, assessed for their health and social care needs. In the presented scenario, the old lady plays this actor. The Older Person must provide information about their health and social care situation, and also receive information such as a summary of their needs and a copy of their care plan. To provide information, the Older Person must undertake assessments. This requires the Older Person to agree with the health and social care professionals on the date/time that the assessments will take place. In addition, the Older Person must understand the procedures clearly, have access to information regarding their care 24 hours every day, and also decide if their information will be shared between the professionals' (and possibly other people) involved in their care. Also, the Older Person must follow the care plan indicated by the health and social care professionals. Therefore, the help of carers (informal-like the daughter- and paid-like the care assistant-) is required.

Nurse: The Nurse performs the overview assessment to the Older Person. To do this, the Nurse must contact the Older Person and arrange a meeting. After performing the assessment the Nurse identifies the care needs of the Older Person, and according to those needs she provides referrals. Also the Nurse must ask for the older person's consent in order to share information with others who may be involved in the care of the Older Person. She generates a care plan and produces a copy of it for the Older Person. In addition, the Nurse informs everyone involved (taking into account the consent of the Older Person) about the care plan and the condition of the Older Person. The Nurse is also responsible for regular review of the care plan.

General Practitioner: The General Practitioner performs the contact assessment, provides referrals to the Nurse to perform an overview (or any different kind she/he thinks is appropriate) assessment, and provides the older person's contact information. In addition, the General Practitioner receives alerts and information

regarding the Older Person, such as the care plan, possible referrals, and updates of the care plan.

Social Worker: The Social Worker receives referrals (indicating the problems occurred) and the actions to be performed, and also information about the Older Person such as contact information and a copy of the care plan. According to the referrals, the Social Worker identifies the needs of the Older Person and takes actions. The Social Worker is usually responsible for identifying a suitable care assistant (if necessary) and also dealing with benefits problems that the Older Person might have. After identifying particular problems the Social Worker provides referrals, informs the other professionals involved in the care of the Older Person and updates the care plan. In addition, the Social Worker manages the care assistant.

Secondary Care Professional: Secondary care professionals (or specialists) undertake assessment and care following referral by primary care professionals. Some secondary care professionals such as community psychiatric nurses work at the interface between primary and secondary care. During the single assessment process, secondary care professionals, usually, do specialist and comprehensive assessments. In the presented scenario, the Occupational Therapist plays this role. The Occupational Therapist receives referrals from the Nurse along with the contact and overview assessment information of the Older Person. She performs a specialist assessment and identifies specialist needs of the Older Person. According to the identified needs, the Occupational Therapist provides referrals, informs the other professionals involved in the care of the Older Person and also updates the care plan.

Care Assistant: The main aim of the Care Assistant is to help the Older Person with everyday needs. The Care Assistant receives information about the Older Person, such as contact and overview assessment, and updates any of those if necessary by providing to the Nurse possible needs of the Older Person. In addition, she informs the General Practitioner, the Social Worker and the Nurse of any updates regarding the older person's information.

Informal Carer: Informal carers include unpaid family members, friends, and neighbours who help meet older persons' needs for care and support, including

meeting emotional (visiting and support), financial (help with managing bills), domestic (help with shopping) and personal (help with dressing) care needs. In the presented scenario the daughter of the old lady plays this role.

Care Manager: A Care Manager, usually a Social Worker or a Nurse, coordinates the delivery of care to the Older Person and plans the work of the care assistants. In the presented scenario, the Social Worker plays the Care Manager.

Benefits Agency: The Benefits Agency actor represents a financial agency that helps older persons financially.

The dependencies, goals and security constraints of the above actors are modelled in Figure 6-2.

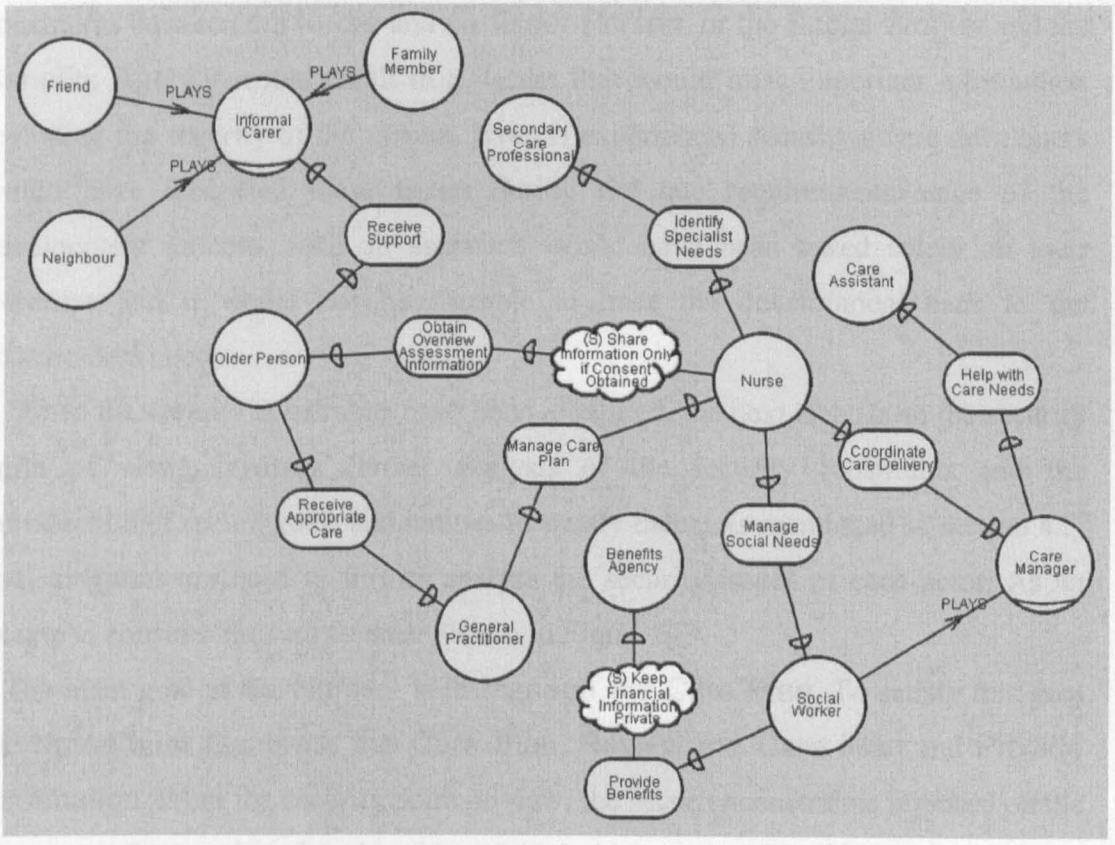


Figure 6-2: The actor diagram

For instance, the Older Person depends on the General Practitioner to Receive Appropriate Care and on the Informal Carer to Receive Support. On the other hand, the Nurse depends on the Secondary Care Professional to Identify Specialist Needs, on the Care Manager to Coordinate Care Delivery, on the Social Worker to Identify Social Needs and on the Older Person to Obtain

Overview Assessment Information. However, one of the most important and delicate matters for the Older Person is the privacy of their personal medical information and the sharing of it. Therefore, the Older Person imposes a security constraint (share information only if consent is Obtained) on the Nurse for the Obtain Overview Assessment Information dependency to be valid. In addition, the Social Worker imposes a security constraint (Keep Financial Information Private) on the Benefits Agency for the Provide Benefits dependency to be valid.

Modelling the security constraints of the individual actors allows developers to model the security requirements of the system according to the real security needs of its stakeholders. In the presented analysis, the lack of identifying the security constraints between the Nurse and the Older Person, or the Social Worker and the Benefits Agency would result in a design that would miss important information regarding the security of the system. Even if experienced security aware developers would have identified these issues during the late requirements stage of the development process, such an approach would have been based solely on their expertise and it would not be possible to trace the development back to the stakeholders needs.

When the security constraints have been identified, the next step (from the security point of view) involves further analysis of the security constraints and the introduction of secure goals and entities to satisfy them. As mentioned in section 4.3, goal diagrams are used to further analyse the security issues of each actor. As an example, consider the Nurse actor shown in Figure 6-3.

The main goal of the Nurse[21] is to Manage the Care Plan. To satisfy this goal the Nurse must Generate the Care Plan, Review the Care Plan and Provide Information. From the security point of view, the security constraints imposed on the Nurse are furthered analysed by identifying which goals of the Nurse they restrict. As mentioned in section 4.3.2.1 the assignment of a security constraint to a goal is indicated using a constraint analysis link (a link that has the "restricts" tag). For example, the Share Information only if Consent Obtained security constraint

[21] To keep the complexity of the figure as minimum as possible, an asterisk * has been used to indicate that the same actor, goal, or entity has been modelled more than once in the figure.

imposed to the Nurse by the Older Person (see Figure 6-2) restricts the Share Older Person Information goal of the Nurse. For the Nurse to satisfy this constraint, a secure goal is introduced Obtain Older Person Consent.

Furthermore, the analysis indicates that the Use of eSAP will enable the Nurse actor to work more efficiently, with less effort, convenient and faster. However, the security reference diagram presented in Figure 6-1 indicates that Authorisation is required for the eSAP system (in order to help towards the Privacy security feature).

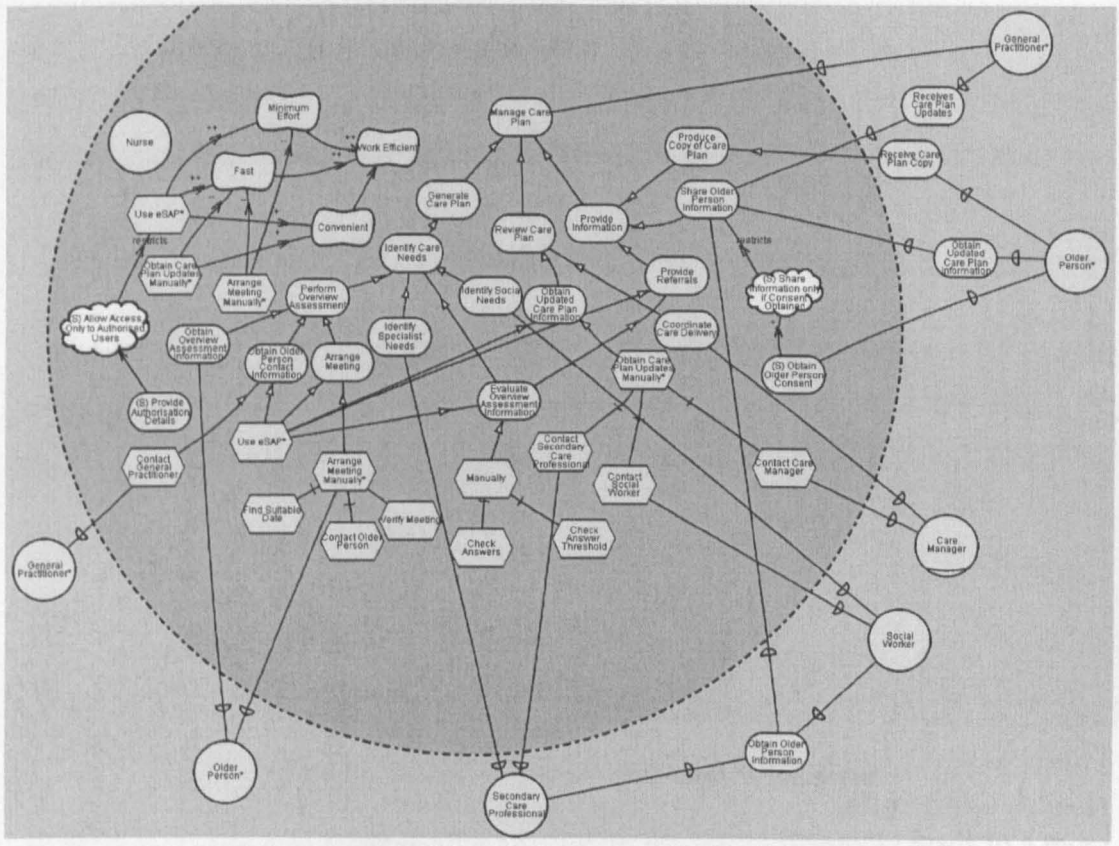


Figure 6-3: Goal diagram for the Nurse actor

Therefore, the security constraint Allow Access Only to Authorised Users, which restricts the Use eSAP task, is imposed to the Nurse actor. To help towards the satisfaction of the imposed security constraint the secure goal Provide Authorisation Details is introduced to the Nurse.

From the Older Person point of view (see Figure 6-4), an important security constraint is to keep their information private. To satisfy this constraint the secure goal Restrict Access to Personal Information has been introduced.

In addition, the internal analysis of the Older Person indicates that the Use of eSAP allows Older Person to obtain information easier, faster and at anytime and therefore helps towards the involvement of the Older Person in their care. However, similarly to the Nurse actor, the Use of eSAP imposes a security constraint (Allow Access Only to Authorised Users) to the Older Person. To satisfy this security constraint the Provide Authorisation Details goal is introduced to the Older Person actor.

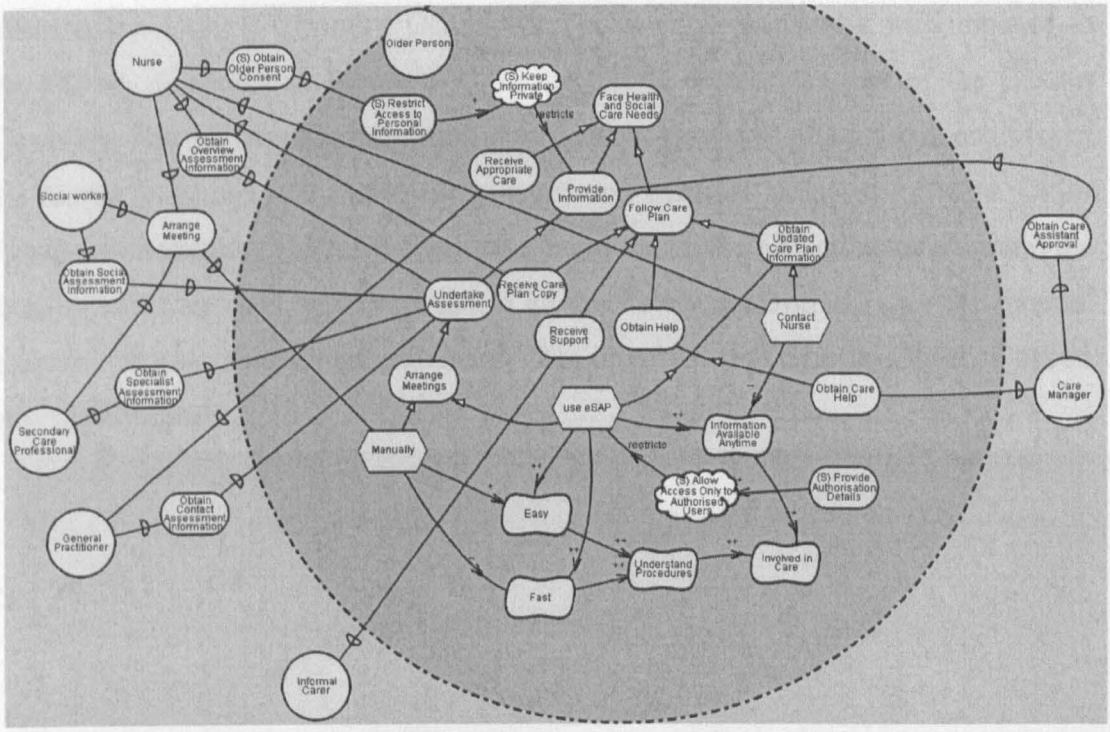


Figure 6-4: Goal diagram of the Older Person actor

Modelling security constraints when analysing the actors internally leads to a more precise definition of the *why* of the system security, and this subsequently helps to verify *how* the final security implementation of the system matches the stakeholders' real needs.

When all the actors have been further analysed[22], the actor diagram is refined, as shown in Figure 6-5, and any possible new dependencies identified during the internal actors' analysis are modelled.

[22] Goal diagrams for the rest of the actors of the eSAP system identified during the early requirements analysis are provided on Appendix B.

This is important since during the actors' internal analysis it is possible that new goals are discovered, which the actors might not be able to satisfy by themselves. Therefore, new dependencies are introduced to enable an actor to delegate to another actor the goals that cannot accomplish on their own. From the security point of view, refining an actor's goals and dependencies could result in the redefinition of the security constraints imposed to particular dependencies or the addition of new security constraints. As an example, consider the security constraint Share Information Only if Consent Obtained. This security constraint was imposed to the Nurse, as shown in Figure 6-3, by the Older Person as part of the Obtain Overview Assessment Information dependency. However, the internal analysis of the Nurse indicated that this security constraint restricts in fact the Share Older Person Information goal of the Nurse. Therefore, in the refined actor diagram, the security constraint has been imposed to all the newly discovered (after the internal analysis of the actors) dependencies that involve the Share Older Person Information goal.

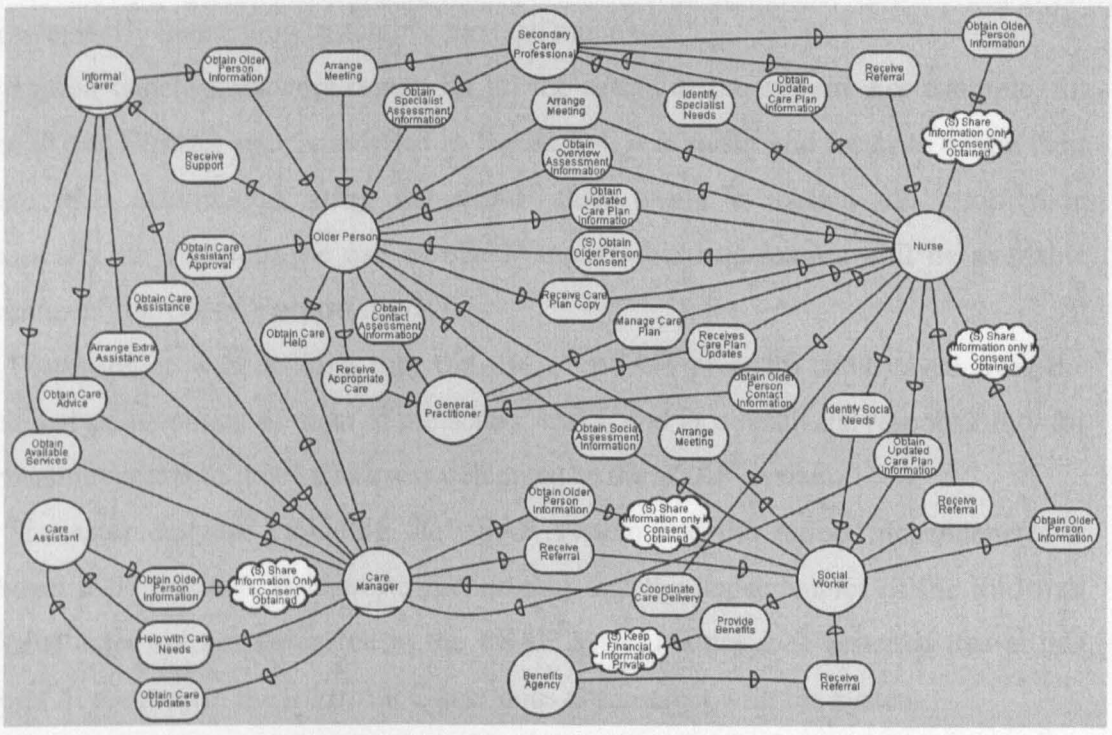


Figure 6-5: Refined actor diagram

6.3.2 Late requirements analysis

As described in section 3.3, during the late requirements analysis the system-to-be is introduced as one or more actors who have a number of dependencies with the other actors.

Therefore, the eSAP system has been introduced as another actor that receives the responsibility for the fulfilment of some of the goals identified during the early requirements analysis for the actors of the system. In other words, some goals that the actors of the system cannot fulfil or are better fulfilled by the eSAP system are delegated to the eSAP System. For example, during the Nurse analysis, modelled in Figure 6-3, it was identified that the Nurse can achieve some of the goals either manually or by using the electronic single assessment process (eSAP) system. Consider for example, the Arrange Meeting goal of the Nurse actor. This can be fulfilled either by the task Use eSAP or by the task Arrange Meeting Manually. However, the analysis, presented in Figure 6-3, showed that using the eSAP system the Nurse would be able to work more efficiently, with less effort, faster and more conveniently than trying to achieve the task manually.

Similar conclusions were drawn for all the actors of the system. For example, for the Older Person actor, modelled in Figure 6-4, it is easier and faster to obtain their care plan information using the eSAP than trying to obtain the information manually. In addition, the use of eSAP means that information will be available whenever the Older Person needs it.

Therefore, it was decided that the use of eSAP provides advantages over the manual achievement of most of the actors' tasks, and as a result the responsibility for the achievement of those tasks was delegated to the eSAP system.

The actor diagram including the eSAP system and the refined dependencies is shown in Figure 6-6. It is worth mentioning that the dependencies of the Informal Carer actor are not delegated to the eSAP system, since it is assumed that at this point of the project the Informal Carer does not interact with the system.

Since dependencies are delegated from the actors to the eSAP system, possible security constraints regarding those dependencies are also delegated.

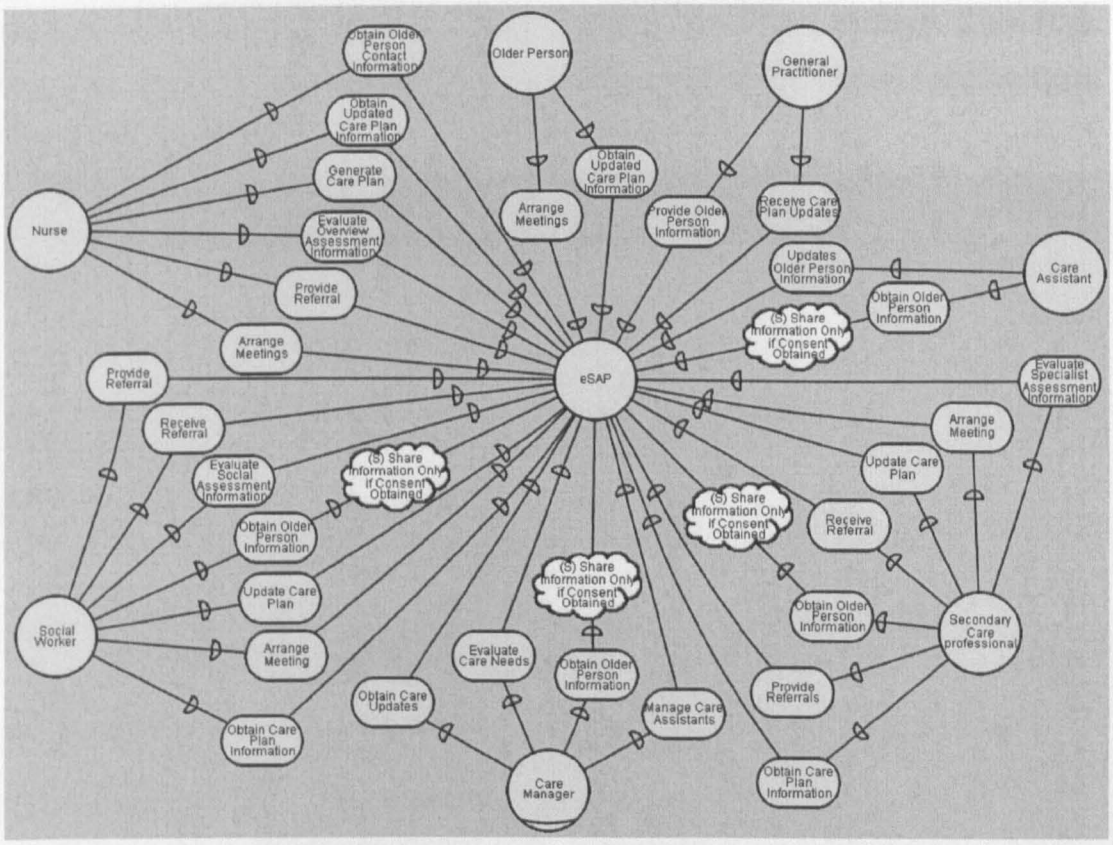


Figure 6-6: Actor diagram including the eSAP actor

For example, before the introduction of the eSAP system, the Social Worker was depending on the Nurse to Obtain Older Person Information. However, this secure dependency involves the security constraint (restricting the Nurse) Share Information Only if Consent Obtained. With the introduction of the eSAP system, the Social Worker actor depends on the eSAP to Obtain Older Person Information, therefore the eSAP becomes responsible for satisfying the Share Information Only if Consent Obtained security constraint that is delegated together with the secure dependency.

To satisfy all the delegated dependencies, the main goal of the eSAP system has been identified as to Automate Care. By performing a means-end analysis, presented in Figure 6-7, it was identified that for the eSAP System to fulfil the Automate Care goal, the following sub-goals must be accomplished: Assist with Assessment Procedures, Provide Older Person Information, Manage Care Plans and Schedule Meetings. Each of those sub-goals can be furthered analysed employing means-end analysis. For example, the Manage Care Plans goal can be

accomplished with the fulfilment of the Generate Care Plan, Manage Care Plan Updates, Provide Care Plan Information, Manage Referrals and Identify Care Assistants sub-goals.

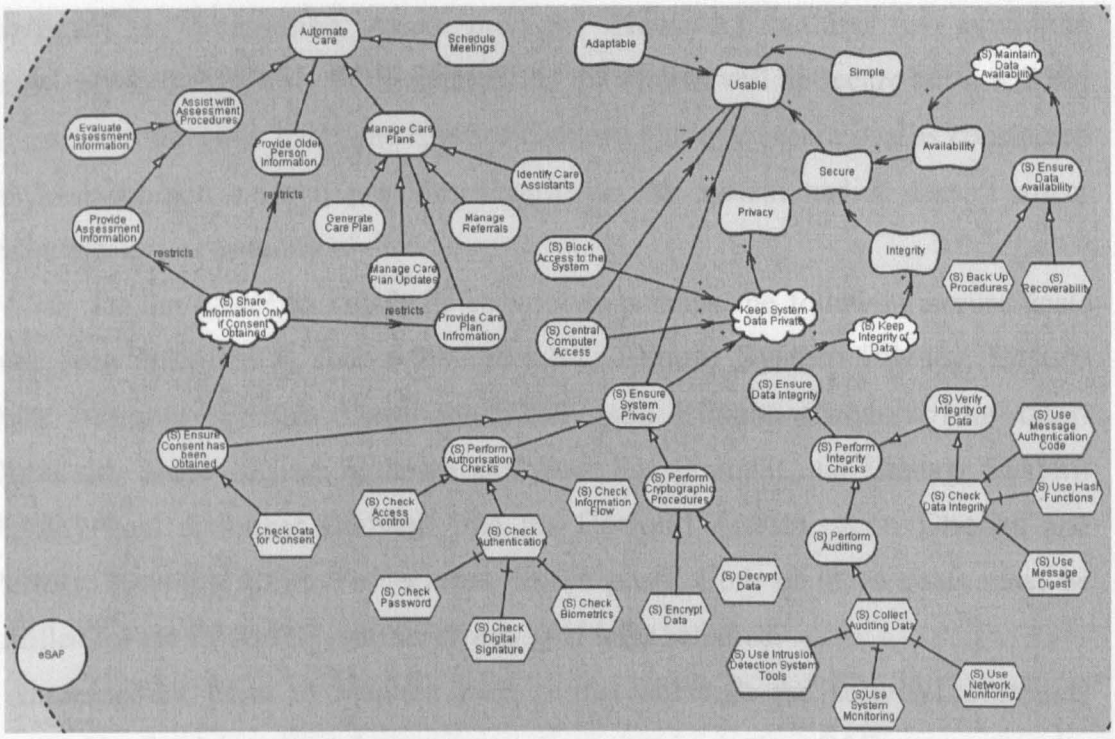


Figure 6-7: Goal diagram for the eSAP actor

An important issue at this point is to check whether the goals assigned in the eSAP system satisfy all the goals delegated to the system by the other actors. Thirty (30) goals were delegated to the eSAP system as shown in Figure 6-6. From these goals, fifteen of them are satisfied by the Manage Care Plans goal (and its sub-goals), six of them are satisfied by the Provide Older Person Information goal, five are satisfied by the Assist with Assessment Procedures goal (and its sub-goals), and four of them are satisfied by the Schedule Meetings goal.

From the security point of view, and taking into consideration the security reference diagram there are three main constraints imposed, by the desired security features of the system, Privacy, Integrity and Availability, to the eSAP's main goal. These are Keep System Data Private, Keep Integrity of the Data and Maintain Data Availability. In addition, the eSAP system must satisfy the Share Information Only if Consent Obtained security constraint imposed to the eSAP by the secure dependencies delegated by the other actors.

Each of these secure constraints can be satisfied with the aid of one or more secure goals. For example, the Keep System Data Private security constraint can be fulfilled by blocking access to the system, by allowing access only from a central computer, or by ensuring system privacy. However, the first two contribute negatively to the usability of the system, i.e. the system will be secure but it will not be used. On the other hand, the Ensure System Privacy secure goal is considered the best solution since it provides security to the system and it doesn't affect (dramatically) its usability.

Thus, for the eSAP to satisfy its security constraints the following secure goals have been identified as shown in Figure 6-7: Ensure System Privacy, Ensure Data Integrity, Ensure Data Availability and Ensure Consent has been Obtained. These can be furthered analysed. For example, the Ensure System Privacy goal is further analysed into the Perform Authorisation Checks and Perform Cryptographic Procedures secure goals. Both of those goals must be fulfilled for the Ensure System Privacy goal to be satisfied.

An important point to mention here is that although the security constraints imposed by the delegation of some secure dependencies to the eSAP system actually restrict particular goals/ tasks of the system, the security constraints imposed with the aid of the security reference diagram actually help, without restricting, towards the achievement of the system's secure goals.

6.3.3 Architectural design

As mentioned in section 5.6, during the architectural design stage the architectural style of the multiagent system is defined with respect to the system's security requirements and according to the analysis technique presented in section 5.2 for selecting among alternative architectural styles.

In this research, for the eSAP system, two architectural styles are considered. A hierarchical style –*client/server* - and a mobile code style -*mobile agents*. The decision to consider these two styles took place because the client/server is the most frequently encountered of the architectural styles for network-based applications, whereas mobile agents form a growing and quite different architectural style. In the client/server style, a node is acting as a server that represents a process that provides

services to other nodes, which act as clients. The server listens for requests upon the offered services. The basic form of client/server does not constrain how the application state is partitioned between client and server components. Client/server architectural style is also referred to by the mechanisms used for the connector implementation such as Remote Procedure Call (RPC). RPC is appropriate for client/server architectural styles since the client can issue a request and wait for the server's response before continuing its own processing. On the other side, in mobile agents style, mobility is used in order to dynamically change the distance between the processing and source of data or destination of results. The computational component is moved to the remote site, along with its state, the code it needs and possibly some data required to perform the task.

As shown in Figure 6-8, each of the two styles satisfies differently each of the non-functional requirements of the system. For instance, the mobile agents style allows more scalable applications (weight 0.8) [23], because of the dynamic deployment of the mobile code.

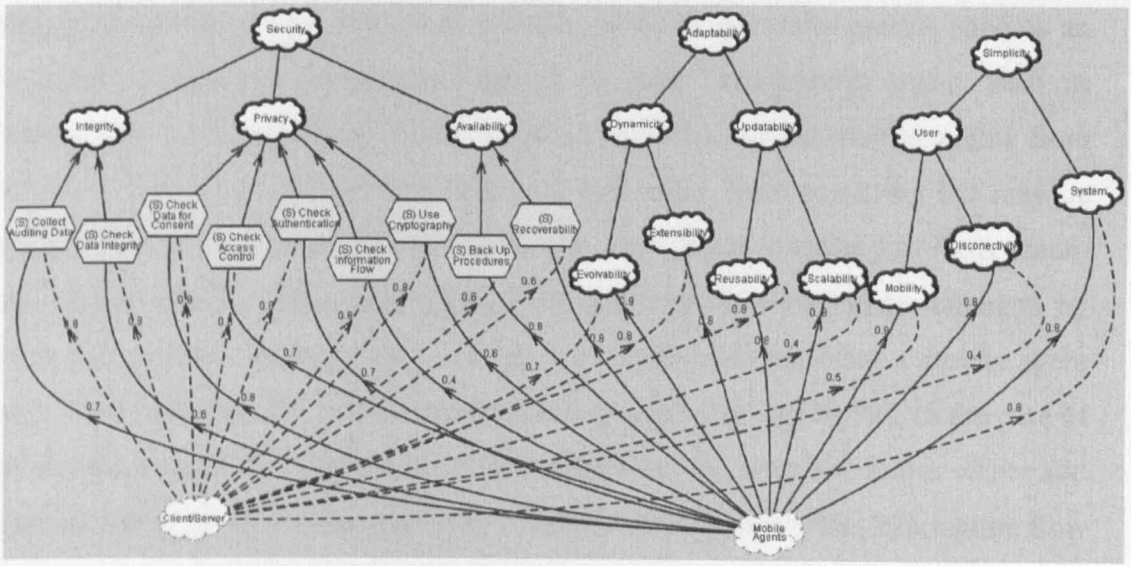


Figure 6-8: Client/Server versus Mobile Agents architectural styles

Consider, for instance, that the Nurse actor wishes to access a large number of medical information (Older Person's care plan), filtered according to the content. In

[23] The weights of the contribution links reported in Figure 6-8, of each architectural style to the different non-functional requirements of the system, have been assigned after reviewing different studies, evaluations, and comparisons involving the architectural styles.

the (pure) client/server architectural style (weight 0.4), the Nurse would access the server data and all the retrieved medical information would be transferred to the client. Then the filtering would be performed at the Nurse site.

On the other hand, in the mobile agents architectural style, such a filtering can be performed in the server site, where redundant information can be identified early and therefore it is not transferred to the client. Therefore, this approach is more scalable since the required filtering is distributed and can be performed close to the information sources.

In the eSAP system, the security of the system is one of the most important factors and it is the criterion that will guide the selection process, in this thesis, for the appropriate architectural style. As derived from the analysis of the eSAP, presented in Figure 6-7, security is decomposed to privacy, integrity and availability.

As concluded from the analysis presented in Figure 6-8, the client/server style satisfies more the privacy requirements of the system than the mobile agents style. This is mainly because mobility is involved in the mobile agents style. Therefore, although protection of a server from mobile agents, or generally mobile code, is an evolution of security mechanisms applied in other architectural styles, such as client/server; the mechanisms focused on the protection of the mobile agents from the server cannot, so far, prevent malicious behaviour from occurring but may be able to detect it [Jan00]. Consider for example, the Check Information Flow secure task of the eSAP. The information flow property is more easily damaged by employing mobile agents (weight 0.4) since possible platforms that a mobile agent could visit might expose sensitive information from the agent [Jan99]. In the case of the client/server style (weight 0.8) sensitive information is stored in the server and existing well-proven security measures could be taken to satisfy the information flow attribute.

On the other hand, the mobile agents style satisfies more, than the client/server style, the availability requirements of the system. Consider for example the recoverability secure task of the eSAP. The Mobile agents style contributes with a weight of 0.8. This is due to the fact that mobile agents adapt dynamically. Mobile agents can react to changes in their environment and maintain an optimal configuration for solving a particular problem [Lang99].

From the integrity point of view, the client/server style contributes better than the mobile agents style. In the mobile agents style mobility is involved and therefore checking the integrity of the data becomes a more difficult task. This is because mobile agents cannot prevent a malicious agent platform from tampering with their code, state or data, but they can only take measures to detect this tampering [Jan00]. Moreover, in the mobile agent style, the integrity of both the local and remote agent platforms must be checked.

From the above, it can be concluded that the client/server styles contributes more towards the privacy and integrity of the eSAP, whereas the mobile agents style contributes more towards the availability. Since privacy and integrity are more important (in the case of the eSAP) than availability (most of the times, not real-time information is needed), the client/server style has been chosen as the architectural style of the system.

When the architectural style has been chosen, the next step of the architectural design stage aims to decompose the system in order to identify internal actors who will satisfy the system's (secure) goals. In the presented example, the eSAP actor is decomposed, as shown in Figure 6-9, to internal actors and the responsibility for the fulfilment of the eSAP's goals is delegated to these actors.

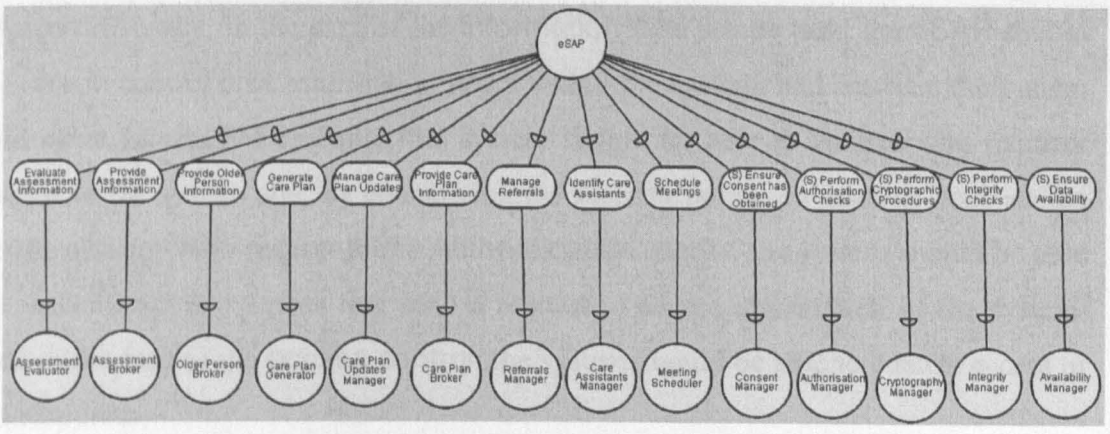


Figure 6-9: Decomposing the eSAP system

For instance, the Evaluate Assessment Information goal is delegated to the Assessment Evaluator, whereas the Provide Assessment Information goal is delegated to the Assessment Broker. In addition, the Older Person Broker and the Consent Manager actors have been introduced to the eSAP system to fulfil the responsibility (identified during the late requirements analysis –see Figure 6-6) of the

eSAP system to satisfy the secure dependency Obtain Older Person Information together with the Share Information Only if Consent Obtained security constraint.

However, the new introduced actors must be furthered analysed and their dependencies with the other (existing and new) actors must be furthered investigated. Such an analysis is important since it helps developers to identify dependencies between new and existing actors, introduce new actors to the system-to-be and, as a result of this, refine the goals of the system or even possibly introduce new goals to the system, which would be very hard to identify otherwise.

As mentioned in section 3.8.2 with respect to security the identification of some of the actors is a difficult, especially for developers with minimum knowledge of security, task. To help developers this research has developed, as described in section 5.3, a security pattern language. Security patterns can greatly help to identify the required actors in a structured manner that does not put in danger the security of the system by providing a solution customised to the problem.

For example, from the internal analysis, presented in Figure 6-7, of the eSAP it was concluded that Information Flow, Authentication and Access Control checks must be performed in order for the eSAP system to satisfy the secure goal Ensure System Privacy. In the case of the information flow secure task, the eSAP should be able to control how information flows within the system, and between the system and other actors. For example, the system should be able to control who requires access to the system and, by considering the security policy, to grant or deny access to the system. With respect to the Authentication checks, the system should be able to authenticate any agents that send a request to access information of the system, and in the case of the access control, the system should be able to control access to its resources.

The proposed pattern language can be used to fulfil the above-mentioned secure goals of the eSAP system. Consider, for example, three of the patterns presented in section 5.3.2.1. The AGENCY GUARD pattern can be used to check grant/deny access to the system according to the security policy, the AGENT AUTHENTICATOR pattern can be used to provide authentication checks and the ACCESS CONTROLER pattern to perform access control checks. The use of

these patterns not only satisfies the fulfilment of the secure goals of the system but also guarantees the validity of the solution.

To apply a pattern, the developer must carefully consider the problem to be solved and the consequences that the application of each particular pattern will have on the system. Figure 6-10 shows a possible use of the AGENCY GUARD, AGENT AUTHENTICATOR and ACCESS CONTROLER patterns in the eSAP system.

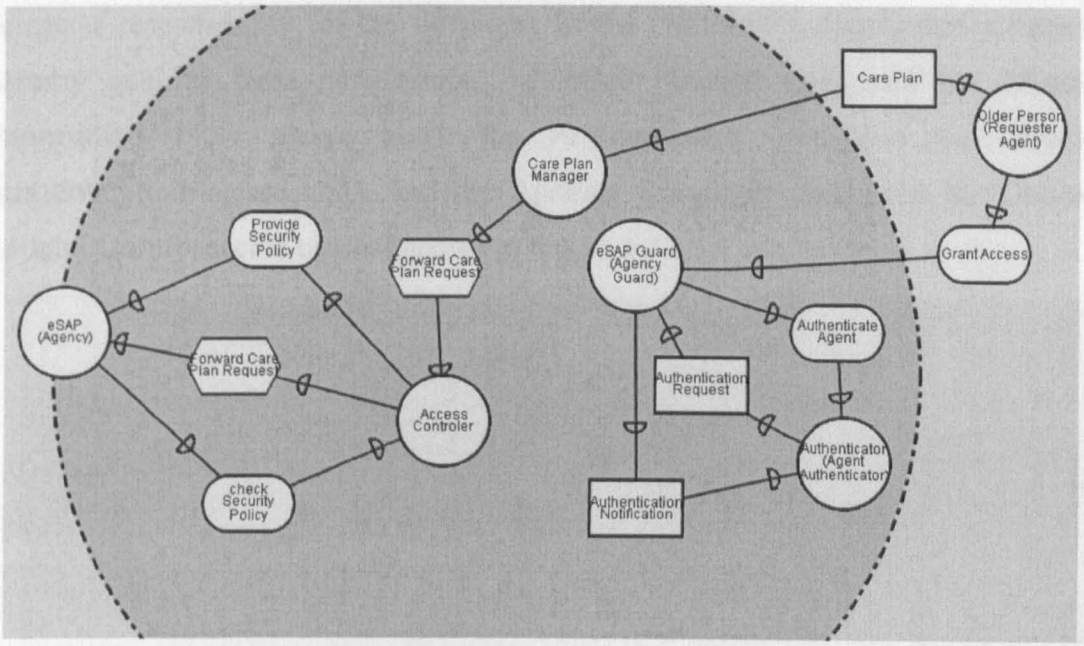


Figure 6-10: Using the AGENCY GUARD, the AGENT AUTHENTICATOR and the ACCESS CONTROLLER patterns in the development of the eSAP

In particular it shows how the secure goals Check Information Flow (problem), Check Authentication (problem) and Check Access Control (problem) can be satisfied. The AGENCY GUARD satisfies the goal by providing a single non-bypassable point of access to the system (solution), the AGENT AUTHENTICATOR satisfies the goal by authenticating each agent that tries to access the system (solution) and the ACCESS CONTROLER controls access to the resources of the system (solution). The use of the patterns helps developers to delegate the responsibilities of particular system security goals to particular actors defined by the patterns. In addition, the developer knows the consequences that each pattern introduces to the eSAP system.

The application of the AGENCY GUARD means that only the AGENCY GUARD must be tested for correct enforcement of the agency's security policy

(consequence), the application of the AGENT AUTHENTICATOR means that during implementation only the AGENT AUTHENTICATOR must be checked for assurance (consequence), whereas the application of the ACCESS CONTROLER means that different policies can be used for accessing different resources (consequence).

Therefore, as derived from the application of the pattern language, the eSAP delegates responsibility for the fulfilment of the Perform Authorisation Checks security goal to three new actors, the eSAP Guard (delegated the Check Information Flow secure task), the Authenticator (delegated the Check Authentication secure task), and the Access Controller (delegated the Check Access Control secure task) as shown in Figure 6-11.

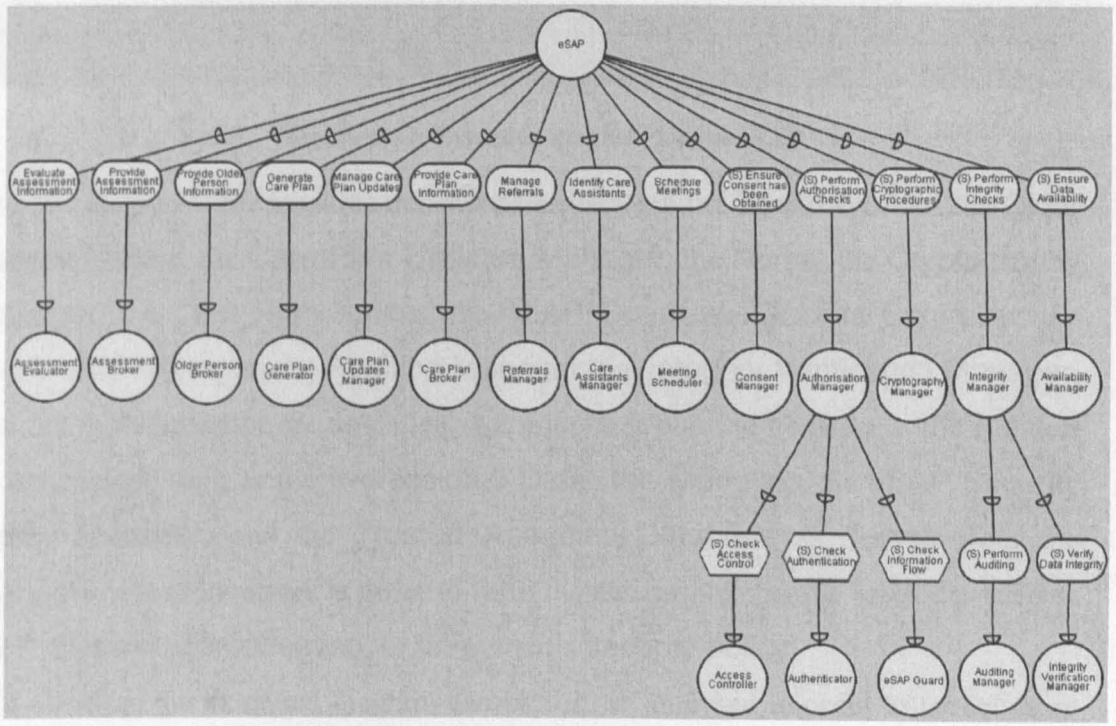


Figure 6-11: Decomposition of the authorisation and integrity managers

In addition, the Tropos methodology introduces extended actor diagrams, in which the new actors and their dependencies with the other actors are presented. As an example, consider the extended diagram depicted in Figure 6-12.

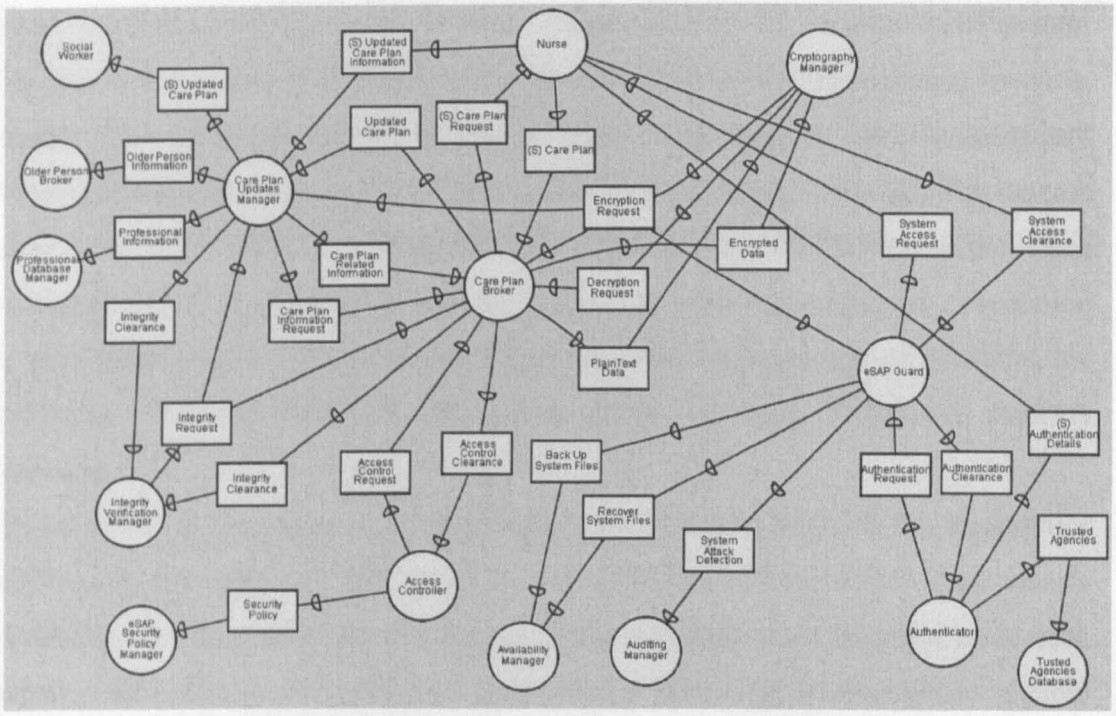


Figure 6-12: Extended diagram for the eSAP

In this diagram[24] the resource dependencies between the Social Worker, the Older Person Broker, the Care Plan Updates Manager, the Nurse, the Cryptography Manager, the Care Plan Broker, the eSAP Guard, the Access Controller, the Availability Manager, the Auditing Manager, the Integrity Verification Manager, and the Authenticator are modelled. An important point to consider is the addition of new actors, such as the Professional Database Manager, the eSAP Security Policy Manager, and the Trusted Agencies Database as derived from the analysis of the other actors in order to fulfil the delivery of specific resources such as the Professional Information, or the system's security policy.

In addition, the extended diagram can be further analysed in order to model more precisely the actors. Consider for instance, the extended diagram with respect to the Assessment Evaluator actor, as depicted in Figure 6-13. The Assessment Evaluator has been delegated the responsibility to satisfy the goal Evaluate Assessment Information. To fulfil this goal, the Assessment Evaluator depends

[24] In order to keep the diagram simple, only some of the actors of the eSAP system have been included in this diagram. Extended diagrams with respect to the other actors can be found in Appendix B.

on two internal actors, the Assessment Analyser and the Evaluation Synthesiser. The first is responsible for obtaining the Assessment Information secure resource, identify the problems of the Older Person according to the Assessment Information and provide the Problems to the Evaluation Synthesiser. The latter is responsible for obtaining the Evaluation Request, and the Problems and providing the Assessment Evaluation secure resource to the actor requesting the information (in the presented analysis to the Social Worker) after considering the Problems, the Available Professionals, the Required Skills and the Proposed Actions resources.

In addition, at this stage, the capabilities identification, in which the capabilities needed by each actor to fulfil their goals and tasks are modelled. Each actor's capabilities can be identified with the aid of the extended actor diagram, since each resource dependency relationship can give place to one or more capabilities triggered by external events. For example the resource Evaluation Request, shown in Figure 6-13, calls for the capability Obtain Evaluation Request for the Evaluation Synthesiser actor and Provide Evaluation Request for the Social Worker actor.

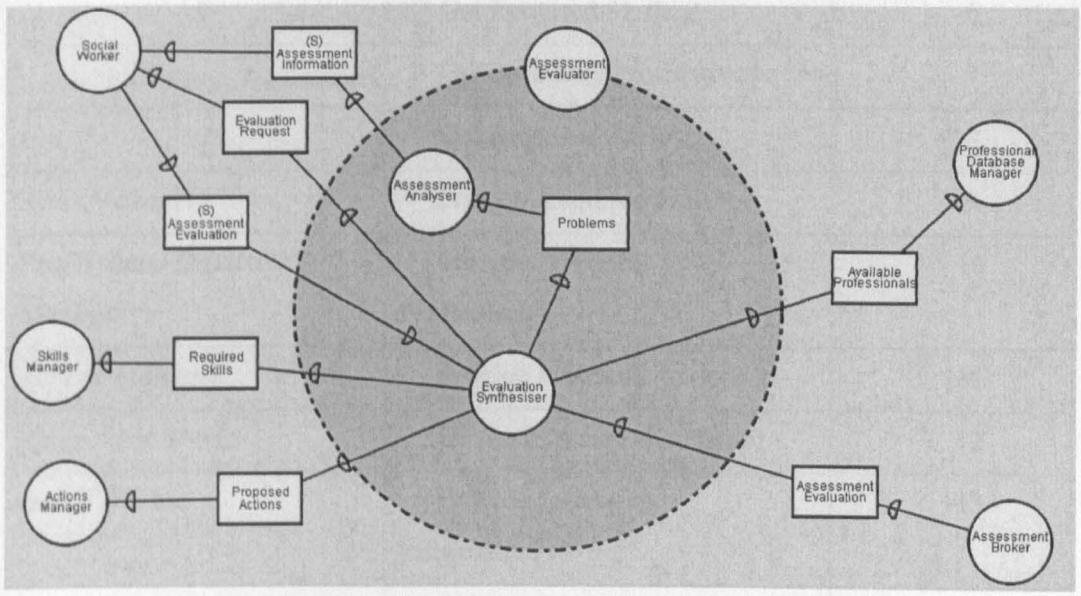


Figure 6-13: Extended actor diagram with respect to the Assessment Evaluator

In addition, secure capabilities are identified taking into account the secure resources of the extended actor diagram. For example, as identified in the early requirements analysis in section 6.3.2, for the eSAP system to satisfy the Ensure System Privacy secure goal, only encrypted data transfers across the network

should be allowed. Therefore, the Assessment Information resource sent from the Social Worker to the Assessment Analyser must be encrypted. Because of this the Social Worker actor should be provided with capabilities to encrypt and decrypt data. Later in the detailed design, each agent's capabilities are further specified and then coded during the implementation phase. Table 6-1 reports the actors of Figure 6-13 and their capabilities as derived from the dependencies that exist between them.

Table 6-1: Actors and their capabilities with respect to the extended diagram of Figure 6-13

Actor	Capability	Capability Id.
Assessment Analyser	Get Assessment Information	1
	Provide Problems	2
Evaluation synthesizer	Get Problems	3
	Get Evaluation Request	4
	Provide Assessment Evaluation	5
	Get Required Skills	6
	Get Available Professionals	7
	Get Proposed Actions	8
Skills Manager	Provide Required Skills	9
Professional Database Manager	Provide Available Professionals	10
Actions Manager	Provide Proposed Actions	11
Assessment Broker	Get Assessment Evaluation	12
Social Worker	Provide Assessment Information	13
	Provide Evaluation Request	14
	Get Assessment Evaluation	15
	Encrypt Data	16
	Decrypt Data	17

When all the actors and their secure capabilities have been identified, the next step of the architectural design is the agents' assignment. During this step a set of agents are defined and each agent is assigned one or more different capabilities, as shown in Table 6-2. The capabilities are assigned according to the actors that the agent represents.

Table 6-2: Agent types and their capabilities

Agent	Capabilities
Assessment Evaluator	1,2,3,4,5,6,7,8
Skills Manager	9
Professional Database Manager	10
Actions Manager	11
Assessment Broker	12
Social Worker	13,14,15,16,17

The last step of the architectural design involves the application of security attack scenarios to the agents of the system. The main aim of these scenarios is to analyse the security of the system by considering the intentions of possible attackers and the secure capabilities that have been assigned to the agents of the system and provide recommendations to improve the system's security.

The security reference diagram plays an important part during this step since it helps to identify threats to the security features of the system. As derived from the analysis of the eSAP system, the three main security features are privacy, integrity and availability. According to Stallings [Sta99], the following categories of attacks can be identified that can endanger the above security features.

1. **Interception**, in which an unauthorised party, such as a person, a program or a computer, gains access to an asset. This is an attack on privacy.
2. **Modification**, in which an unauthorised party not only gains party to but also tampers with an asset. This is an attack on integrity.
3. **Interruption**, in which an asset of the system is destroyed or becomes unavailable or unusable. This is an attack on availability.

Therefore, scenarios that involve each of these categories of attacks will be considered.

6.3.3.1 Interception Scenario

Lets consider an interception attack scenario in which a possible attacker wishes to attack the privacy of the system, in other words to obtain information such as assessment information or a care plan. As identified in the analysis of the security reference diagram, social engineering, password sniffing and eavesdropping are the main threats to the privacy of the system.

Therefore, the attacker's main goal can be decomposed to Read Data and Get Access to the System sub-goals as shown Figure 6-14. The first sub-goal involves the attacker trying to read the data that it is transmitted to and from the eSAP system, whereas the second sub-goal involves the attacker trying to break into the system and gain access to it.

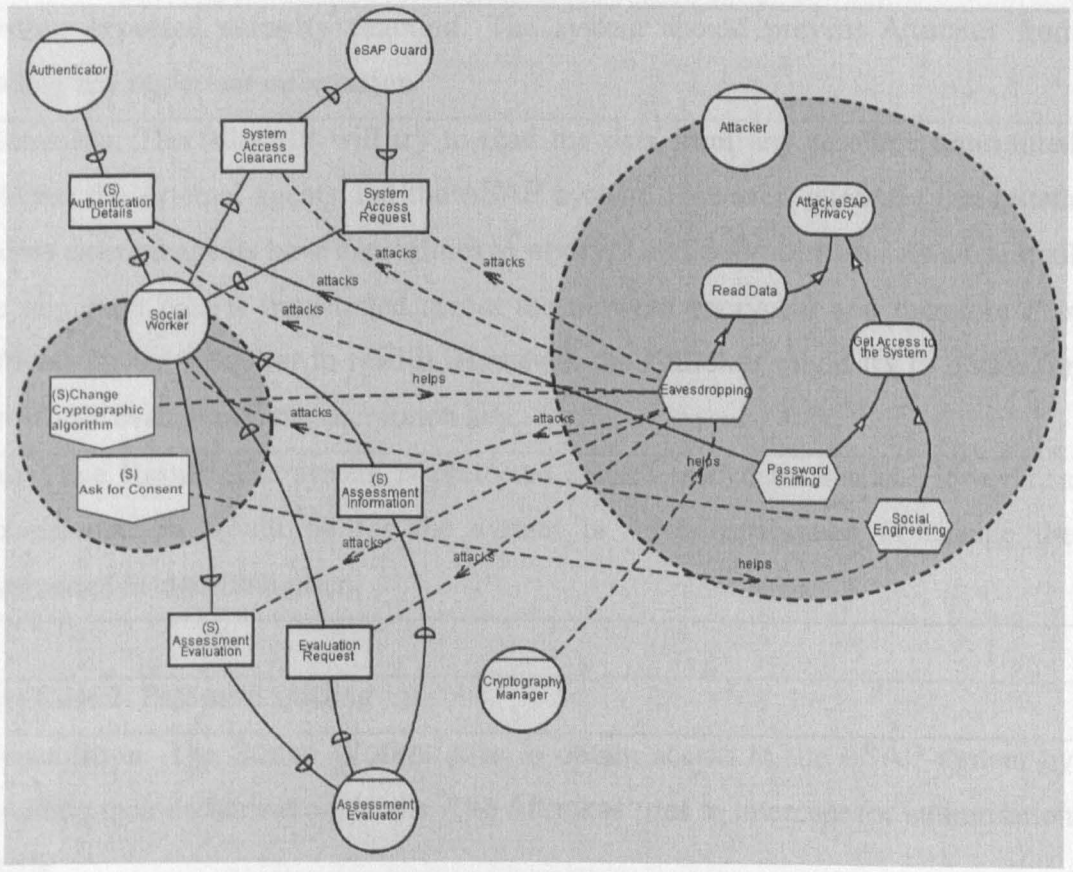


Figure 6-14: Interception attacks scenario

With respect to Figure 6-13, to accomplish the first sub-goal the Attacker should try to read the data transferred between the Social Worker and the eSAP system's actors such as the Assessment Evaluator and the Authenticator. To accomplish the second sub-goal, the Attacker might use password sniffing or social engineering. In the first case, the Attacker scans all the resources that flow in the network looking for passwords whereas in the case of social engineering, the Attacker tries to deceive the Social Worker in order to obtain valuable information, such as their authorisation details that will allow them to gain access to the system.

Therefore, for the presented attack scenario, the reaction of the system should be tested against three test cases, read data, password sniffing and social engineering.

Test Case 1: read data
Precondition: The Social Worker actor tries to obtain an assessment evaluation. The Attacker tries to read the transmitted data.
System expected security reaction: The system should prevent Attacker from reading any important information.
Discussion: The Attacker will try to read the data from any resource transmitted between the external agents and the eSAP system. However, curerntly the system and its external agents have capabilities to encrypt and decrypt data. As a result all the important data is transmitted across the network encrypted and therefore it is difficult for the Attacker to read it. However, the Attacker might try to obtain (or sometimes even guess) the encryption key.
Test Case Result: The system is protected against read data attacks. However, a recommendation would be for the system to have capabilities to change the cryptographic algorithm often.

Test Case 2: Password sniffing
Precondition: The Social Worker tries to obtain access to the eSAP system by providing their authorisation details. The Attacker tries to intercept the authorisation details.
System expected security reaction: prevent the Attacker from obtaining users' passwords

Discussion: the main target of the Attacker would be all the resource transmitions between the Social Worker and the eSAP system that contain any kind of authorisation details. Although authorisation details are enrypted, this is not enough since password sniffing takes place from a compromised computer belonging to the network. As a result, the Attacker is able to decrypt any message. A good technique to defend against password sniffing is to use one-time-passwords. A one-time-password is a password that is valid for only one use. After this use, it is not longer valid, and so even if the Attacker obtains such a password it is useless. However, the users must be able to gain access to the system more than once. This can be accomplished with what is commonly known as a password list. Each time a user tries to access the system they provide a different password from a list of passwords.
Test Case Result: Currently the system fails to adeqautely protect against password sniffing attacks. For the eSAP system to be able to react in a password sniffing attack, the external agents of the system (such as the Nurse, the Social Worker, the Older Person) must be provided with capabilities to provide passwords from a password list.

Test Case 3: Social engineering
Precondition: The Attacker tries to obtain system information directly from the Social Worker.
System expected security reaction: help towards the prevention of social engineering.
Discussion: The Attacker will try to deceive any external agents (such as the Social Worker in the presented scenario) into giving any confidential, private or privileged information. It is worth mentioning that the Attacker will not directly ask for this information but they will try to gain the trust of the agents and then exploit this trust.
Test Case Result: Currently the system helps towards the prevention of social engineering by requesting consent for any information to be shared. However, this alone does not guarantee the successful prevention against social engineering. A primary defence measurement against software engineering is security awareness

training. Good resistance training will help to prevent agents from being persuaded to give information away.

6.3.3.2 Modification Scenario

The modification scenario involves an Attacker that wishes to attack the integrity of the eSAP system. As identified in the analysis of the security reference diagram, three main threats are involved in this kind of attack, cryptographic attacks, care plan changing and viruses.

Therefore, the Attacker's main goal, Attack eSAP Integrity, can be decomposed to Modify Content of Messages, Change Values in Data Files, and Alter Programs to Perform Differently as shown in Figure 6-15.

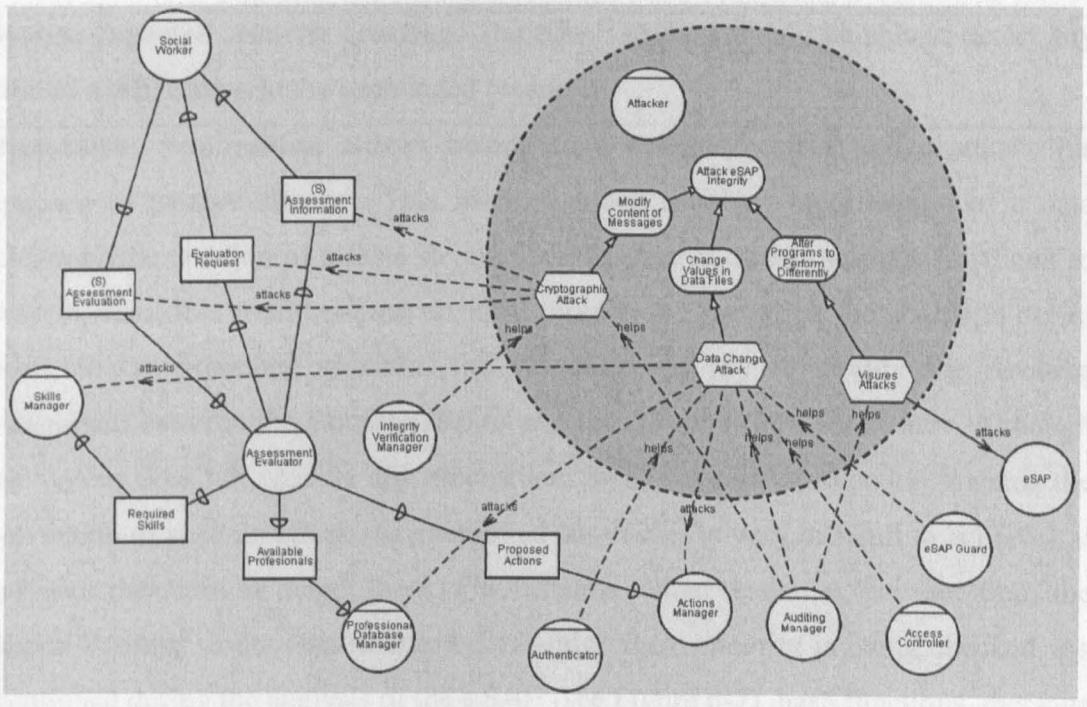


Figure 6-15: Modification attacks scenario

The first sub-goal involves the Attacker trying to modify the content of any messages transmitted over the network. To fulfil this goal, the Attacker might try to employ cryptographic attacks to any resource transmitted between any external actors and the eSAP system. The second sub-goal indicates the Attacker trying to change the values in data files of the system. The fulfilment of this goal can be satisfied by means of changing the data of resources stored in the eSAP system. The third sub-goal indicates the attempt of the Attacker to alter a program so it performs

differently. Mainly this can be achieved using viruses that can alter the behaviour of specific programs (agents) in order to enable the attacker to gain access to the system or to system's information.

As an example, consider the scenario in which the Social Worker wishes to obtain an Assessment Evaluation. Three main test cases are identified, cryptographic attacks, data changing attacks and viruses attacks as shown in Figure 6-15.

Test Case 1: cryptographic attacks
Precondition: The Social Worker actor tries to obtain an assessment evaluation. The Attacker tries to modify the content of the messages/resources exchanged between the Social Worker and the Assessment Evaluator.
System expected security reaction: The eSAP system should be able to detect any kind of modification to the exchanged resources.
Discussion: modification attacks belong to a category called active attacks (as opposed to passive attacks). This kind of attack involves modification of a data stream or the creation of a false stream [Sta99]. Active attacks are quite difficult to prevent, since this would require physical protection. Therefore, the goal is to detect them. In the presented scenario, the Attacker will try to modify the resource transmitted between the Social Worker and the Assessment Evaluator. Although the system does not provide any mechanism or any security protection towards the prevention of such an attack (as mentioned above this is very difficult to achieve), it provides measures to detect them. For instance, when resources are sent from the Social Worker to the Assessment Evaluator their integrity is being checked. As mentioned during the analysis of the eSAP (see Figure 6-7), hash functions, message digest and message authentication codes are employed by the eSAP to satisfy the integrity of messages exchanged between the eSAP and external actors.
Test Care Result: The system provides mechanisms to detect any modifications resulting from cryptographic attacks.

Test Case 2: changing data
Precondition: The Attacker tries to change values of data stored in the eSAP system.

System expected security reaction: The system should prevent attacks towards the unauthorised manipulation of its data.
Discussion: The Attacker will try to gain access to the system in order to change values of resources stored in the system. For instance, it might change the name of the General Practitioner allowed to view an Older Person's care plan. Towards this kind of attack, the system basically offers three layers of protection. First of all, only authorised users are allowed access to the system. But even if the Attacker manages to obtain somehow access to the system (through social engineering for example) access control checks are in place to make sure that every authorised user has access only to necessary resources. In addition, auditing tests are performed by the eSAP system. This involves the collection of data relating to the behaviour of authorised users. Then users are observed to determine any sudden changes to their behaviour.
Test Case Results: The system provides mechanisms to protect against possible attacks aiming to change the data of the system.

Test Case 3: Viruses
Precondition: The Attacker tries to change the system behaviour by using some kind of virus.
System expected security reaction: The system should be able to prevent viruses.
Discussion: Viruses consist one of the most sophisticated threats to computer systems. It is quite common for attackers to send viruses to computer systems they want to attack in order to exploit vulnerabilities and change the behaviour of the system. Although many effective countermeasures have been developed for existing types of viruses, many new types of viruses are also developed frequently. An ideal measurement against viruses is prevention. In other words, viruses should not get into the system. However, this is almost impossible to achieve. Therefore, the best approach is to be able to detect, identify and remove a virus. Auditing helps towards the detection of the virus. However, apart from this the eSAP system is not protected against viruses.
Test Case Results: The eSAP system needs to be integrated with an anti-virus program to enable it to effectively detect, identify and remove any possible viruses.

Such a program, which could be another internal agent of the eSAP system, should be able to monitor the system and take effective measurements against any possible viruses.

6.3.3.3 Interruption Scenario

As mentioned above, interruption attacks mainly aim the availability of the system. From an Attacker's point of view, such attacks can be mainly categorised into two main categories, physical attacks and electronic attacks (see Figure 6-16). Physical attacks include any attacks to the infrastructure of the system, whereas electronic attacks involve attacks such as denial of service attacks.

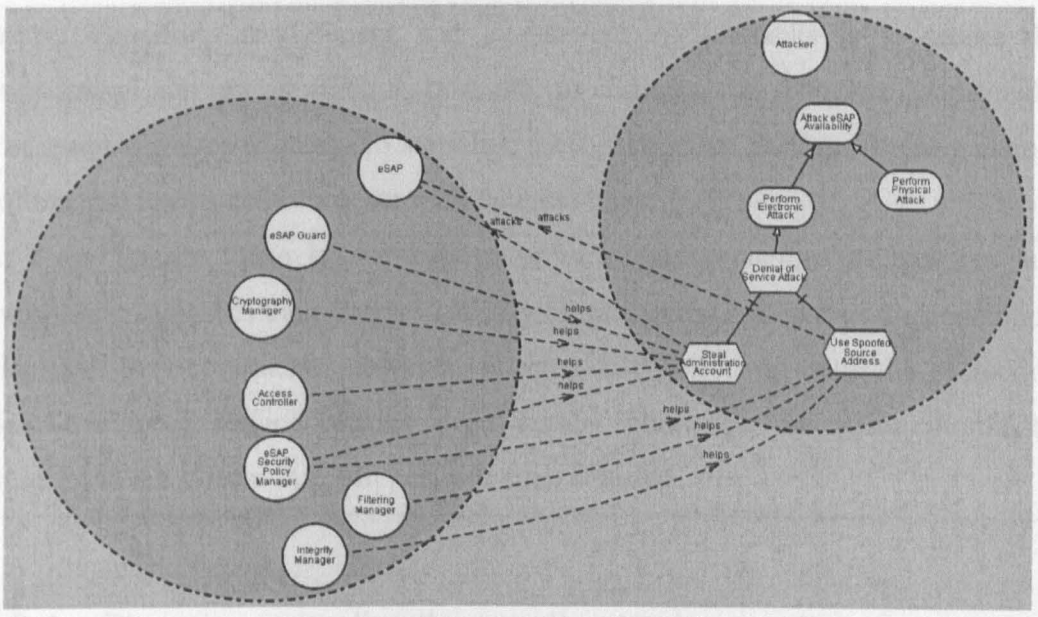


Figure 6-16: Interruption attacks scenario

Therefore, the Attacker's main goal (Attack eSAP Availability) can be decomposed to physical and electronic attacks. Physical attacks involve the cutting of a communication line, or the destruction of a part of the system. On the other hand, one of the most popular electronic attacks to the availability of a system is denial of service attacks. Since physical attacks to the eSAP system are outside the focus of this research project, only a test case involving a denial of service attack is considered.

Test Case: denial of service
Precondition: The Attacker tries to make the eSAP system unavailable by

performing a denial of service attack.
System expected security reaction: the eSAP should be able to detect the attack and recover.
Discussion: During a denial of service attack, the Attacker tries to prevent the normal operation of the communication facilities of the system [Sta99]. Since a denial of service attack is an active attack, the main goal of the eSAP system is to detect the attack and recover from any disruption it may cause as fast as possible. Towards this direction, the agents of the system must have capabilities to operate even if some other agents have become unavailable. Mostly denial of service attacks require from Attackers to steal an administration account of a hose computer in the network. Therefore, an efficient way to prevent such attacks is to secure the administration account as much as possible. In addition, the Attacker might make use of spoofed source address. To stop this, the system must perform filtering mainly when internal agents communicate with external ones.
Test Case Results: The eSAP system provides authorisation mechanisms and therefore helps towards the effective security of the system and in turn the prevention of denial of service attacks. However, filtering is required to make the protection against denial of service attacks even better. Therefore, an agent should be introduced to the system that will perform such filtering.

6.3.3.4 Discussion regarding the security attack scenarios

In order to test the security of the system, three different kind of scenarios were identified involving seven different test cases. By applying these test cases many useful results were obtained about the security of the eSAP system. First of all, it was identified that the system provides enough protection against some of these attacks. Secondly, for the attacks that the system did not provided adequately protection, extra agents and extra secure capabilities were identified and the following modifications took place in the eSAP system.

1. Capabilities were given to the external agents and to the Cryptography Manager to enable them to change the cryptographic algorithm often. The

lack of such capabilities was identified during the read data test case of the interception attack scenario modelled in Figure 6-14.

2. The external agents of the system were given the capability to provide passwords from a password list, and the Authenticator was given capabilities to successfully process such passwords. The lack of such capabilities was identified by the application of the password-sniffing test case of the interception attack scenario.
3. An agent, called Viruses Monitor, is introduced to the system to monitor the eSAP and take effective measurements against any possible viruses. The lack of such an agent was identified by the application of the viruses test case of the modification attack scenario presented in Figure 6-15.
4. An agent, called Filter Agent, is introduced to the system to filter the eSAP in order to help towards the protection of denial of service attacks. The lack of such an agent was identified by the application of the denial of service test case of the interruption security attack scenario presented in Figure 6-16.

Table 6-3 illustrates the agents of the eSAP system as derived from the analysis presented in the previous sections together with the agents identified from the analysis of the security attack scenarios. The capabilities of each of these agents can be found in Appendix B.

Table 6-3: The agents of the eSAP System

Assessment Analyser	Care Plan Updates Manager	CA Information Collector	Access Controller	Cryptography Manager
Assessment Synthesiser	Care Plan Broker	CA Information Provider	Authenticator	Skills Manager
Assessment Broker	Referral Provider	Assistant Proposer	eSAP Guard	Professional Database Manager
Older Person Broker	Referral Constructor	Meeting synthesiser	Auditing Manager	Actions Manager

Care Plan Generator	Referrals Database	Meeting Notifier	Integrity Verification Manager	eSAP Sec. Policy Manager
Care Plan Format Database	Assistants Database Manager	Consent Manager	Availability Manager	Trusted Agencies Manager
Filter Agent	Viruses Monitor	Social Worker	Older Person	Nurse
General Practitioner	Care Assistant	Care Manager	Secondary Care Professional	

6.3.4 *Detailed design*

When the attack scenarios stage has been completed and the capabilities of the agents have been refined to provide as much security from possible attacks as possible, the next step involves the specification of the system's components.

The important consideration, from the security point of view, during the detailed design stage is the specification of the system components by taking into account their secure capabilities.

For instance, a partial class diagram related to the Meeting Scheduler is shown in Figure 6-17. The important consideration regarding security is that the eSAP Guard must check the security privileges of any possible Meeting Initiator or Meeting Participant before allowing them to interact with the Meeting Scheduler.

Moreover, as mentioned in section 3.5, to represent the capabilities of the agents, Tropos employs capability diagrams to model a capability from the viewpoint of a specific agent and plan diagrams to specify each node of the capability diagram. The same diagrams are used to represent the secure capabilities of the agents.

Consider for example, the Receive Care Plan Request secure capability of the Care Plan Broker. This can be depicted as shown in Figure 6-18.

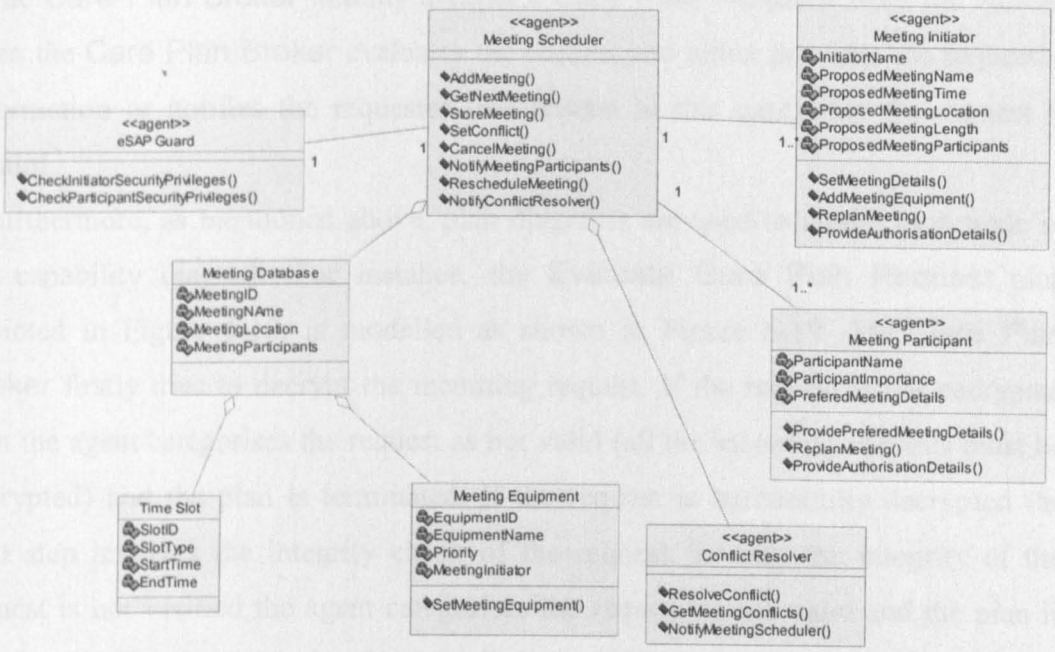


Figure 6-17: Partial Class Diagram with respect to the Meeting Scheduler

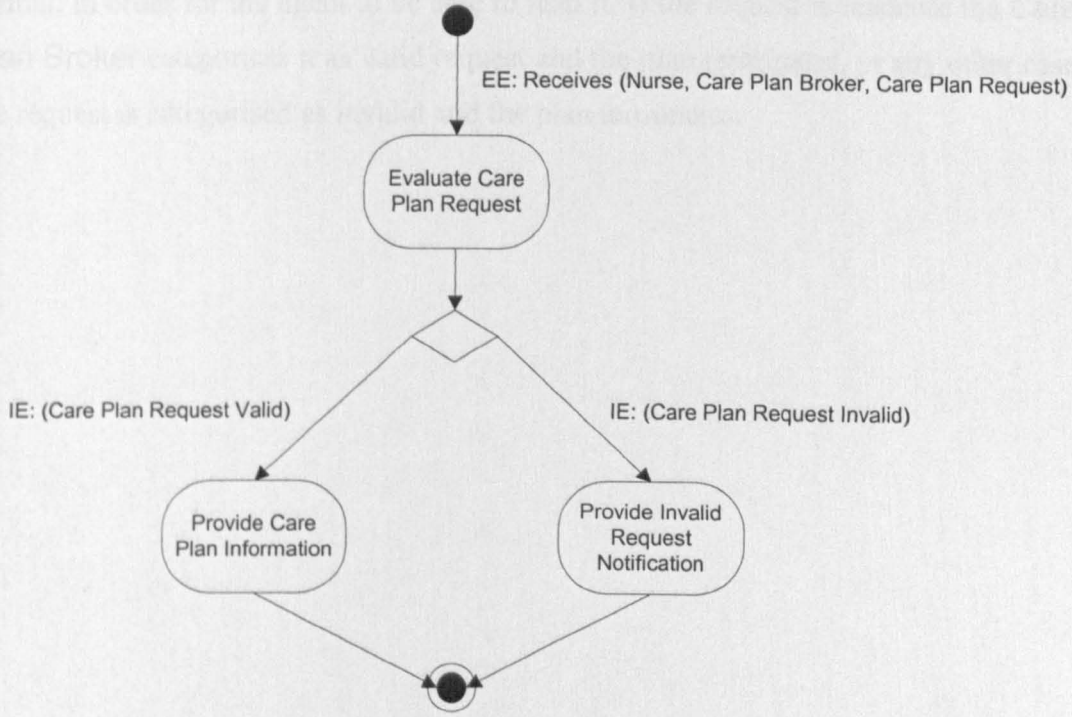


Figure 6-18: Capability diagram for the Receive Care Plan Request **capability of the** Care Plan Broker **agent**

The Care Plan Broker initially accepts a Care Plan Request from the Nurse. Then the Care Plan Broker evaluates the request and either provides the requested information or notifies the requester (the Nurse in this case) that the request is invalid.

Furthermore, as mentioned above, plan diagrams are used to model each node of the capability diagram. For instance, the Evaluate Care Plan Request plan depicted in Figure 6-18, is modelled as shown in Figure 6-19. The Care Plan Broker firstly tries to decrypt the incoming request. If the request is not encrypted then the agent categorises the request as not valid (all the incoming requests must be encrypted) and the plan is terminated. If the request is successfully decrypted the next step involves the integrity check of the request. In case the integrity of the request is not verified the agent categorises the request as not valid and the plan is terminated. The last step involves reading the request in order for the agent to respond to it. It must be noticed that every incoming request follows a specific format, in order for the agent to be able to read it. If the request is readable the Care Plan Broker categorises it as valid request and the plan terminates, in any other case the request is categorised as invalid and the plan terminates.

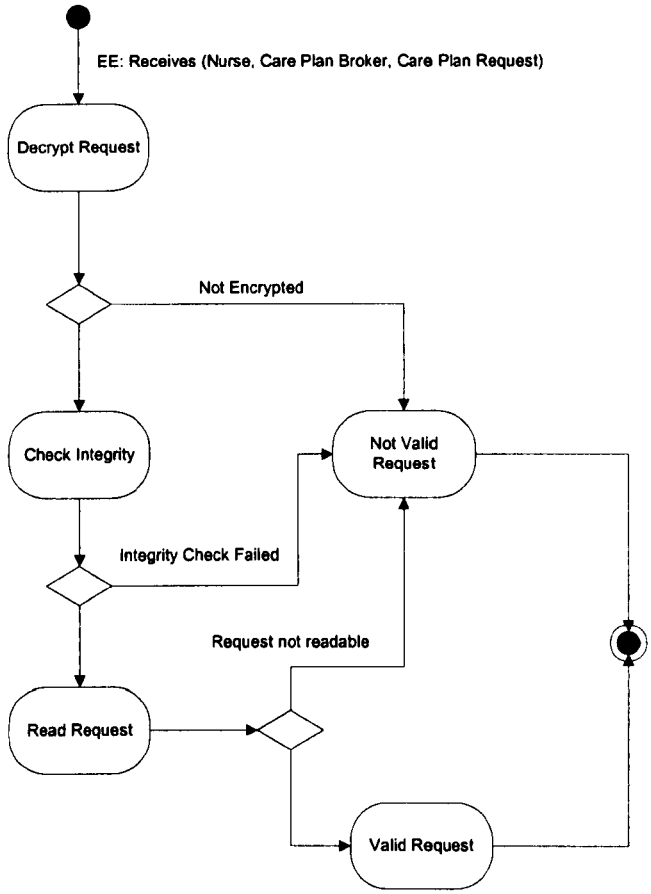


Figure 6-19: Plan diagram for the evaluate care plan request plan

In addition, agent interaction diagrams are used to model the interactions of the agents. Consider for example, the interactions that take place when the Social Worker tries to obtain access to the system.

The Social Worker sends an encrypted message to the eSAP Guard requesting access to the system. The eSAP Guard forwards the request to the Cryptography Manager for decryption. After the Cryptography Manager decrypts the request it forwards it plain text to the eSAP Guard. Then the eSAP Guard checks the authentication privileges of the Social Worker with the aid of the Authenticator. Then the Authenticator requests from the Social Worker to send their authentication details. When the Authenticator receives the authentication details of the Social Worker either provides an authentication clearance or rejects the authentication of the Social Worker. After the authentication clearance has been granted, the eSAP Guard provides system access clearance to the Social Worker.

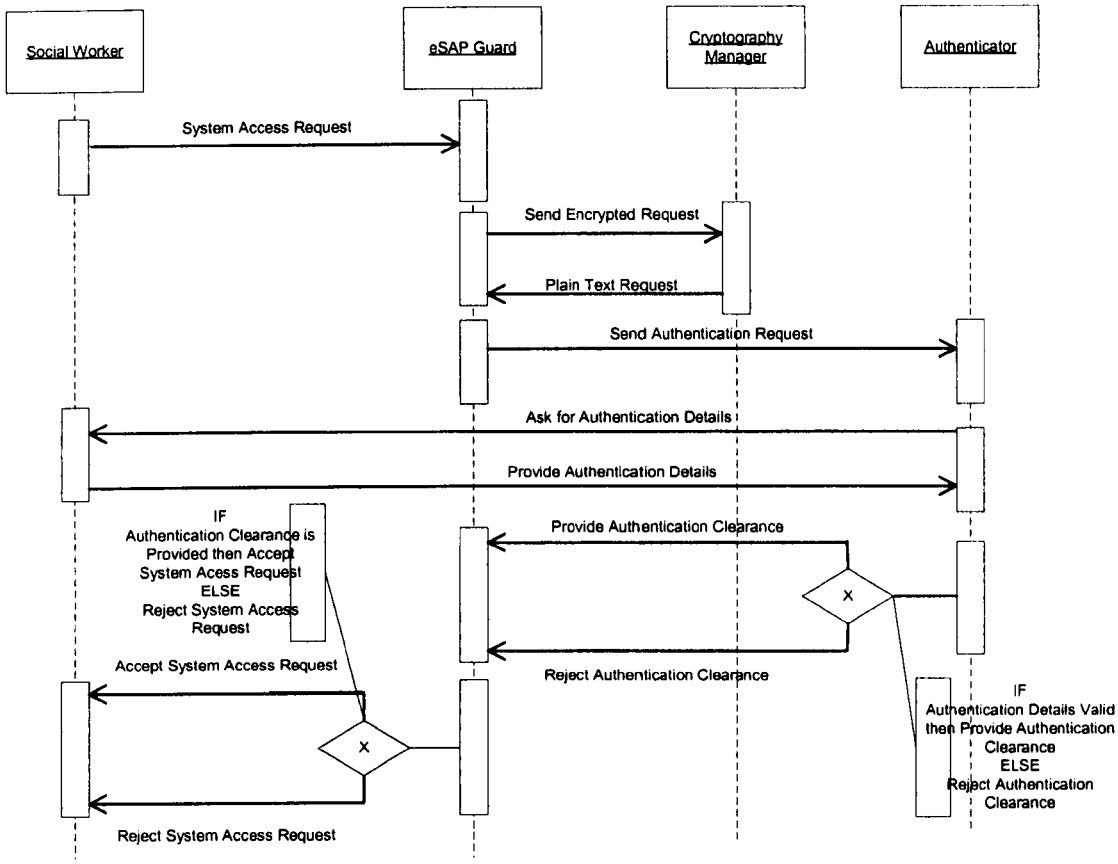


Figure 6-20: Interaction diagram for the Social Worker system access clearance

6.4 *DISCUSSION REGARDING THE SECURITY EXTENSIONS*

Chapters 4 and 5 introduced security-oriented extensions to the Tropos methodology to enable it to model security issues throughout the whole development process of a multiagent system. In addition, the previous section of this chapter presented how the proposed security-oriented extensions can be employed for the development of a real-life health and social care information system for older people. The main aim of this section is to provide a critical discussion/evaluation of the presented approach.

6.4.1 How the proposed security-oriented approach helps towards the development of secure multiagent systems

By modelling the security constraints of the individual actors, developers are able to model the security requirements of the system according to the real security needs of its stakeholders. For example, during the eSAP system analysis, the lack of

identifying the security constraints between the Nurse and the Older Person, or the Social Worker and the Benefits Agency would result in a design that would miss important information regarding the security of the system.

Furthermore, by imposing security constraints, and differentiate between security-related and non-security-related goals and entities, developers can define together security and other functional and non-functional requirements of a multiagent system and at the same time provide a clear distinction between them. This distinction helps towards the detection of possible conflicts between security and other requirements, and therefore allows developers to analyse those conflicts and propose possible ways towards a design that will overcome them, leading to the development of a more secure system.

Moreover, the introduction of the security reference diagram allows the identification of desired security requirements very early in the development stages, and helps to propagate them throughout the development stages. This introduces a security-oriented paradigm to the software engineering process. In addition, the security reference diagram helps to discover, by taking into account the security requirements of the system, possible security constraints that must be introduced to the system-to-be as well as possible security mechanisms that contribute to the satisfaction of the security constraints that are imposed on the system. This and the fact that security expert developers can expand the security reference diagram, provides security novices with a valuable reference point when considering security issues during the development of multiagent systems.

In addition, the transformations developed for the construction of the security reference diagram, the modelling activities for the security-related concepts, the identification of security-related stages, and the successful integration of the approach within the development stages of the Tropos methodology contribute towards a clear and well guided security-oriented process that allows even non-security oriented developers to consider security in their design.

Moreover, the integration of the pattern language within the development stages of the Tropos offers a suitable solution for the development of secure agent based systems. The modelling language of the methodology provides a framework that forms the base for the development and application of the pattern language, since the

patterns of the language and the relationships between them are expressed and described by employing concepts from the methodology ontology. Furthermore, the integration of the pattern language within the Tropos methodology allows novice security developers to reason about the consequences a particular design will have on their system, and therefore develop a design that will satisfy the security requirements of the system.

In addition, the technique for selecting amongst different architectural styles and its integration within the Tropos methodology allows the explicit definition of the technique and allows developers to evaluate and select between different designs according to the system's security requirements. This, in turn, allows developers to analyse security requirements and base design solutions on this analysis. On the other hand, the introduction of security attack scenarios to test the system's response to potential attacks and the definition of a set of consistency rules to check the security-oriented approach allows developers to test the developed security solution and also check the consistency of the development process.

In addition, the proposed approach employs the same concepts and notations throughout the development process. Therefore, the developer is not concerned with the "translation" of a concept from one stage to another. This allows a uniform development, leading to a better definition of the multiagent system.

6.4.2 The key features of the proposed approach

An important key feature of the proposed approach is that the security requirements of the system can be traced back to the requirements of the stakeholders. For example, in the eSAP analysis presented in the previous section, the secure task of the eSAP system Check Data for Consent can be traced back to the early requirements analysis of the Older Person actor and their secure goal to Restrict Access to Personal Information.

In addition, the concept of constraints is a natural extension of the Tropos methodology. This, together with the minimum changes on the notation, allows developers familiar with Tropos to easily model security issues. The notation used, an S within brackets, to indicate security related concepts can be easily adopted or

ignored if the developers wish, and also it can be easily extended to indicate other non-functional requirements.

On the other hand, the iterative nature of the presented security-oriented process allows the re-definition of security requirements in different levels and as a result it provides better integration with the modelling of the system's functionality, whereas the consideration of the organisational environment for the modelling of security issues facilitates the understanding of the security needs in terms of the security policy and the real security needs of the stakeholders.

The usage of the attack scenarios provide developers the ability to realistically check how the developed system will react to possible security attacks. This, in turn, allows developers to re-consider particular system functions with respect to security until the system under development satisfies all the security requirements.

6.5 SUMMARY

The main aim of this chapter was to illustrate how the proposed security-oriented extensions can be applied in the development of a real life information system. To fulfil this aim, this chapter described how the extended Tropos methodology was employed for the development of the electronic single assessment process (eSAP) system, a health and social care information system for the effective care of older people. The presented illustration analytically described how the extensions are applied in each step of the development process.

Although, at each step of the development process discussions took place to indicate the strengths and the important points of the security-oriented approach, the chapter also provided a critical discussion that indicates how the proposed approach helps in the development of more secure systems. In addition, this chapter described the key features of the proposed security-oriented approach as derived from the application of the approach to the development of the electronic single assessment process system.

The following chapter concludes this thesis.

CHAPTER 7 CONCLUSIONS

The preceding three chapters have described an extended version of the Tropos methodology to enable developers of multiagent systems to consider security issues throughout the development of a multiagent system using the same concepts and notations throughout. Chapter 4 described security-oriented extensions to the concepts and the modelling activities of the Tropos methodology, whereas chapter 5 described a security oriented process and it outlined how the Tropos methodology stages can be refined to include the proposed security-oriented process. Finally, chapter 6 described how the proposed approach can be employed in the development of the electronic single assessment process system and it also provided a critical discussion indicating how the proposed approach helps towards the development of secure multiagent systems.

The final chapter of this thesis starts, in section 7.1, by evaluating whether the presented security-oriented approach meets the minimum set of requirements, set in section 2.3.2.2, that a security oriented process should satisfy. Then, section 7.2 revisits the objectives set at the beginning of this thesis, presented in section 1.4, and discusses whether the developed approach satisfies them. Moreover, the main contributions and the significance of this research are discussed in sections 7.4, and 7.5 respectively. Finally, directions for future work are outlined in section 7.6 and a summary of the thesis is provided in section 7.7.

7.1 DOES THE PROPOSED APPROACH SATISFY THE REQUIREMENTS SET IN SECTION 2.3.2.2?

Section 2.3.2.2 identified eleven requirements that a security-oriented approach must satisfy. The following paragraphs discuss how the presented approach has successfully met them.

- The introduction of the concept of security constraints and the extension of the existing Tropos concepts with security in mind greatly helps towards the satisfaction of requirements 4, 6 and 7 described in section 2.3.2.2 since it

allows for a systematic approach towards the modelling of security requirements.

- The differentiation between security-related and non-security-related goals and entities allows developers to define together security and other functional and non-functional requirements of a multiagent system and at the same time provide a clear distinction between them. This introduces a security-oriented paradigm to the software engineering process and therefore contributes towards the satisfaction of requirement 4 of section 2.3.2.2.
- The introduction of the security diagram, and the fact that the diagram can be expanded by security aware developers, provides security novices with a valuable reference point when considering security issues during the development of multiagent systems and as a result such an approach contributes towards the satisfaction of requirement 1 of section 2.3.2.2.
- The adoption of current Tropos concepts for the development of the diagram allows the usage of the same concepts and notations throughout the development process and therefore contributes towards requirement 2 of section 2.3.2.2.
- The introduction of the security oriented process and the successful integration of the approach within the development stages of the Tropos methodology contribute towards a clear and well guided security-oriented process that allows even non-security developers to consider security in their design and therefore satisfies requirements 3 and 4 of section 2.3.2.2.
- The technique for selecting amongst different architectural styles allows developers to analyse security requirements and evaluate and select a design according to the system's real security requirements and as a result it satisfies requirement 10 of section 2.3.2.2.
- The integration of the pattern language within the development stages of Tropos offers a suitable solution for the development of secure agent-based systems and contributes towards requirements 3, 8, and 9 of section 2.3.2.2
- The introduction of security attack scenarios to test the system's response to potential attacks and the definition of a set of consistency rules to check the

security-oriented approach allow developers to test the developed security solution, and therefore satisfies requirement 11 of section 2.3.2.2, and it also checks the consistency of the development process, and as a result satisfies requirement 5 of the section 2.3.2.2.

From the above, it is concluded that the proposed approach satisfies all the requirements presented in section 2.3.2.2.

7.2 DISCUSSION ON HOW THE RESEARCH OBJECTIVES WERE MET

An important issue when concluding a project is to identify whether the work that took place satisfies the objectives (and therefore the aim) set at the beginning of the project. Thus, the following paragraphs review one by one the objectives set at the beginning (although some of them have been evolved during the project) of this project, and present a discussion on each one of these.

Objective 1: Identify problems of integrating security and systems engineering and provide a set of minimum requirements necessary for a security oriented process.

To satisfy the first part of this objective, this research identified seven main problems (presented in section 2.3.2.1) associated with the integration of security and software engineering. Amongst others, these include the involvement of non-security experts in the development of multiagent systems that require knowledge of security, the diversity of concepts used by software engineers and security engineers, and the difficulty to move from a set of security requirements to a design that satisfies these requirements. In addition, to satisfy the second part of the above objective, this research identified a set of minimum requirements (eleven requirements) that a security oriented process should demonstrate (presented in section 2.3.2.2). These requirements indicate (amongst other things) that the process should allow novice security developers to successfully consider security issues, employ the same concepts and notations throughout the whole development lifecycle, and be integrated within a methodology.

Objective 2: Extend the concepts and notations of an existing agent oriented software engineering methodology with respect to security modelling.

This research project reviewed different agent oriented software engineering methodologies and it identified their strengths and limitations with respect to security modelling, as presented in section 2.4.1.1. Then it identified the Tropos methodology as a suitable methodology for the security oriented extension. This decision was mainly based on the fact that the Tropos spans in all the development stages using the same concepts, it is easily extensible and also it is more security aware than other methodologies. In addition, the Tropos methodology is well integrated with other approaches, such as the Agent UML, in which some security work has taken place, and therefore existing work can be considered and incorporated within the proposed approach.

However, the Tropos methodology demonstrated a number of significant limitations in its concepts and notations with respect to security modelling (section 3.8), such as the inadequacy of the soft-goal concept to model security issues, and the lack of definition of current concepts with security in mind. Therefore to satisfy this objective, the concept of a security constraint was introduced, and existing concepts such as goals, dependencies and capabilities were defined with security in mind. For example, the concept of secure goal was identified, which represents the strategic interests of an actor with respect to security. Moreover, notation for the security related concepts was added at the methodology's notation. To keep the notation simple and easy to understand, security related concepts are modelled by adding an S within brackets "(S)" on the root concepts of the Tropos methodology. As it was described, in chapter 4, this allows minimum modifications in the notation of the methodology and therefore makes it easy to understand, especially by developers familiar to the Tropos methodology, and also it allows developers to introduce other concepts, such as performance, easily in the current notation.

Objective 3: Develop a clear, well guided process of integrating security and systems engineering throughout the software development process of multiagent systems, using the same concepts and notations throughout the process.

To satisfy this objective this research developed, as presented in chapter 5, a security oriented process that is divided into four main activities. (1) The identification of security requirements of a multiagent system; (2) the selection amongst alternative architectural styles for the system to be according to the identified security requirements; (3) the development of a design that satisfies the security requirements of the system; and (4) the attack testing of the multiagent system under development.

Security requirements are identified by employing the modelling activities developed by this research such as security reference diagram construction, security constraints and secure entities modelling, presented in Chapter 4. The process of identifying the security requirements of the system is basically one of analysing the security needs of the stakeholders and the system in terms of security constraints imposed on the system and its stakeholders and identifying secure goals and entities that guarantee the satisfaction of the security constraints as described in section 5.1.

When the security requirements have been identified the second step of the process involves the selection of an architectural style for the system according to the specified security requirements. For this reason, this research has developed, as presented in section 5.2 an analysis technique, which is based on an independent probabilistic model, to enable developers to select among alternative architectural styles using as criteria the non-functional requirements of the multiagent system under development.

To allow the development of a design that satisfies the security requirements, this research proposes in section 5.3 the use of patterns. Towards this direction a pattern language consisting of security patterns for multiagent systems was developed. The use of such a language enables non-security specialists to identify patterns for transforming the security requirements of their system into design, and also be aware of the consequences that each of the applied security patterns introduce to their system. Additionally, since security patterns capture well-proven solutions, it is more likely that the application of security patterns will satisfy the security requirements of the system.

To test the reaction of the system under development to potential security attacks, this research proposed in section 5.4 a technique that is based on the use of scenarios.

A scenario, called Security Attack Scenario, includes enough information about the system and its environment to allow validation of the security requirements. This approach identifies the goals and the intentions of possible attackers, identify through these a set of possible attacks to the system (test cases), and apply these attacks to the system to see how it copes. By analysing the goals and the intentions of the attackers the developer obtains valuable information that helps to understand not only *how* the attacker might attack the system, but also *why* an attacker wants to attack the system. This leads to a better understanding on how possible attacks can be prevented. In addition, the application of a set of identified attacks to the system contributes towards the identification of attacks that the system might not be able to cope with (failed test cases) and this leads to the re-definition of the agents of the system and the addition of new secure capabilities to enable them to protect against these attacks. Moreover, in section 5.5, a set of consistency rules was developed to allow developers to check the consistency of the security-oriented development process.

Objective 4: Integrate the security oriented process within the methodology's development stages.

To satisfy this objective, the Tropos development stages have been refined, in section 5.6, to accommodate the proposed security extensions. During the early and late requirements analysis stage the security requirements are identified. The security reference diagram is constructed and security constraints are imposed to the stakeholders of the system (by other stakeholders). These security constraints are furthered analysed (with the aid of goal diagrams) and secure goals and entities are introduced to the corresponding actors to satisfy them. In addition, security constraints are imposed to the system-to-be (by reference to the security reference diagram) and these constraints are analysed.

In the architectural design, the architectural style of the multiagent system is defined with respect to the system's security requirements and the requirements are transformed into a design with the aid of security patterns. Furthermore, the agents of the system are identified along with their secure capabilities and Security Attack Scenarios are used to test the security of the system under development. Then, at the

detailed design stage, the components identified in the previous development stages are designed with the aid of Agent UML.

Objective 5: Evaluate the proposed solution by applying it for the development of the electronic single assessment process system.

To satisfy this objective, the proposed security related approach was employed for the development of the electronic single assessment process system, a real life complex health and social care information system, and also a critical discussion of the approach was presented in section 6.4 together with the key features of the proposed approach.

Although the presented approach cannot claim that by employing it a totally secure system will be developed[25], the application of the approach in the development of the electronic single assessment process indicated that the proposed approach provides valuable help and allows (even non security aware) developers to consider security issues throughout all the development stages when developing systems with agent orientation in mind.

This is mainly due to two main reasons. Firstly it provides a well guided process that enables even non-security specialists to reason about the security of the system, and secondly it provides a process to check the security of the developed system and redefine it according to a set of security attack scenarios. This leads to the development of a more secure system.

It is worth mentioning that the developed system presented in this chapter corresponds on the scenario identified in section 6.2. As such, the identified set of actors and their (secure) capabilities is not complete with respect to an electronic system to deliver the single assessment process taking into account the whole setting surrounding the single assessment process. However, the presented development provides a very good basis for which a complete design for the single assessment process can be developed. The actors identified in this chapter can be employed and the only difference would be the identification of extra actors and their (secure)

[25] Such a claim would be false and in fact none can claim something like this because, as mentioned earlier (section 2.3.1) in this thesis, there is no such system as a totally secure system.

capabilities required by the system in order to deliver the extra functionality and also the definition of the relationships (dependencies) between the already identified actors and the newly introduced ones.

Although the proposed approach was applied in the development of an information system for the health care domain, it is also applicable to any other information system that demonstrates similar characteristics. However, there are some limitations and the approach is not suitable for any kind of software development. First of all, because the proposed approach is based on the Tropos methodology, it follows the same limitations imposed by the Tropos methodology [Bre02b]. As a result, it is not applicable for the development of embedded software or system software (operating systems for instance) since in such systems there are no identifiable stakeholders.

Moreover, the approach is not suitable for performing specific security related analysis activities, such as check that exchange of data obeys the security levels [Jur01], analyse security requirements at the physical layer [Jur01], and specify and verify security protocols [Ban89, Mea94]. Such activities imply the consideration of a particular implementation and are out of the scope of this work.

7.3 INTEGRATION TO OTHER METHODOLOGIES

As mentioned in chapter 4, the proposed security oriented extensions are mainly divided into two categories. Concepts related and process related extensions. Although, the proposed security concepts and notations have been specifically developed for the Tropos methodology, the proposed security oriented process can be integrated to other methodologies with few modifications. The following paragraphs discuss how the proposed security oriented process could be integrated within the agent oriented software engineering methodologies discussed in chapter 2, i.e. GAIA, MAS-Common KADS, and MaSE.

During the analysis stage of the GAIA, the security reference diagram could be constructing, and the security requirements of the system could be identified taking into consideration the identified roles and their permissions. However, some concepts related extensions would be necessary to allow this process. For instance, although permissions help to model security related permission that the system might have, they fail to model possible security restrictions of the system or the associated

roles. Then during the design stage, the security design pattern language could be employed to help in the aggregation of the roles to agents, and the architectural style of the system could be identified using the proposed process for selecting architectural styles. Finally, security attack scenarios could be developed from the agents, services and acquaintance models.

A more substantial effort would be required to integrate the approach to the MAS-Common KADS methodology. The conceptualisation stage of the methodology is appropriate for the construction of the security reference diagram. However, new security related concepts should be introduced to help the identification of the security requirements of the system, since the current methodology concepts are very limited for this activity. Then, during the agent design phase the security pattern language could be used to help the identification of the most "security" related architecture for each agent, and the information modelled in the agent network design, such as information related to the network facilities sub activity, could be used to construct the security attack scenarios. On the other hand, the methodology does not address the issue of designing the system's architecture and as a result, the selection of and architectural style according to the security requirements of the system proposed activity cannot be integrated in the methodology.

The MaSE methodology starts its requirement analysis by capturing the system's goals. As such, most of the security related concepts, such as security constraints, and secure goals/tasks, proposed by this thesis can be integrated within the methodology to help developers during the security requirements identification activity, which in turn can be integrated within the analysis stage of the MaSE together with the security reference diagram. The security design pattern process could be integrated within the design stage and in particular in the assembling agent classes activity, whereas the selection of the architectural style according to the security requirements of the system could be integrated within the system design activity. Finally, the security attack scenarios could be constructed by obtaining information from the previous stages of the analysis and design.

7.4 RESEARCH CONTRIBUTIONS

This thesis introduced an agent oriented approach in the development of information systems, which considers security issues as an integral part of the whole development process. The Tropos methodology has extended to allow developers to consider security throughout all the development process. As a result, this research advances the current state of the art in agent oriented software engineering in four important ways:

- It identifies limitations of current agent oriented software engineering methodologies with respect to security modelling.
- It points out a set of problems in the integration of security and software engineering, and identifies a set of requirements for a security oriented approach.
- It extends the Tropos methodology, a widely known agent oriented software engineering methodology, with respect to security modelling.
- It employs the extended methodology in the development of a real life health and social care information system.

Therefore, the contributions of this research project can be summarised in the following points:

- It introduces a security-oriented paradigm to the software engineering process using the same concepts and notations throughout the development process.
- It provides a systematic, clear, and well guided approach towards the modelling of security requirements.
- It allows developers to define together security and other (functional and non-functional) requirements of a system and at the same time provide a clear distinction between them. This helps to limit the cases of conflict between security and functional requirements, by identifying them very early in the development process and find ways to overcome them.
- It allows the identification of desired security requirements very early in the development stages, and helps to propagate them throughout the development stages.

- It allows novice security developers to reason about the consequences (with respect to security) a particular design will have on their system, and therefore develop a design that will satisfy the security requirements of the system.
- It allows developers to evaluate and select between different designs according to the system's security requirements.
- It allows developers to test the system's response to potential attacks.

There are also very important contributions of this work, outside the computer science area. This research argued that the software agent paradigm is suitable for developing systems for the health and social care sector, since both of them (agent paradigm and health and social care systems) exhibit a considerable number of mutual characteristics, such as cooperation and share of information. On the other hand, it identified the lack of security modelling as an important consideration for the application of agent oriented software engineering methodologies in the development of health and social care information systems. Therefore by extending current agent oriented software engineering methodologies to help in the development of designed solutions of health and social care systems with security in mind, this research actually introduced a novel approach in developing systems for the health and social care sector.

7.5 SIGNIFICANCE OF THIS RESEARCH

According to the Computer Crime and Security Survey, contacted by the Computer Security Institute (CSI) with the participation of the San Francisco Federal Bureau of Investigation's computer intrusion squad, during 2003 about (90) percent of respondents, mainly large US corporations and US government agencies, detected computer security breaches and seventy - five (75) percent acknowledged financial losses due to those security breaches.

Security vulnerabilities have also been dramatically increased the last few years. According to the CERT Coordination Center[26] while during 1995, 171 vulnerabilities were reported, this number increased to 1993 during the first two quarters of 2003. In

[26] http://www.cert.org/

addition, the last 10 years the number of incidents reported has increased from 1334 (in 1993) to 76,404 (the first two quarters of 2003).

All those figures prove that security is not considered as much as it should. A reason for this is that for software developers, security interferes with features and time to market. Although, security specialists use mathematical security models for the development of secure information systems, these models are very complex and difficult to understand by software engineers without security expertise. However, software engineers have to develop multiagent systems that require security features. Thus, the definition of security requirements is usually considered after the design of the multiagent system. This typically means that security enforcement mechanisms have to be fitted into a pre-existing design therefore leading to serious design challenges that usually translate into software vulnerabilities.

By integrating security and systems engineering, this research provides an alternative security-oriented approach in the development of multiagent systems. Such an approach allows the identification of possible conflicts between security and functional requirements before the actual implementation of the system. This, in turn, enables developers to find ways to overcome these conflicts without rebuilding the system and therefore save valuable industrial time and money. Furthermore, by providing a structured, well understood development process using the same concepts and notations throughout the development stages, this research allows software engineers without security expertise to reason about security when developing a multiagent system.

Moreover, the development of an agent based system to deliver the single assessment process will have a major impact in the health and social care professionals related to the delivery of care to older people in England. By analysing and designing such a system with security in mind, this research work provides the foundation in which a successful (future) implementation can be based on. An agent would be allocated to each professional, and it would be given enough intelligence so that it can negotiate with agents of other professionals to minimise the workload of the professionals and maximise the cooperation required for the efficient care of older people, thus improving the care of older people.

7.6 DIRECTIONS FOR FUTURE WORK

When a project is finished, a number of issues that pose new challenges appear. There are many directions in which the work described in this thesis can be extended to increase the chance of success of the proposed approach.

Modelling trust and ownership is a very important issue, in a multiagent system, and it is closely related to the modelling of security. Recently, Giorgini et al. [Gio03] proposed an enhanced version of the Tropos methodology to allow it to appropriately model trust relationships. Their approach could be integrated together with the approach presented in this thesis to allow a more complete analysis of security relationships that exist in a multiagent system. Since both approaches have been developed and integrated within the Tropos methodology the task of integrating the two approaches is very feasible.

Another interesting area of investigation is the extension of the Tropos formal specification language to include the security related concepts. Formal Tropos complements graphical Tropos by extending the Tropos graphical language into a formal specification language [Fux01, Fux03]. Formal Tropos can be employed to perform a formal analysis of the system and also verify the model of the system by employing formal verification techniques, such as model checking, to allow for an automatic verification of the system properties [Fux01]. Although some work [Mou03d] was initiated, as part of this research, towards the extension of the Formal Tropos concepts to include security, it was later decided for this research to focus only on the graphical Tropos. However, the initial work can be the basis for a full extension of the formal Tropos to consider security issues.

In addition, an interesting area for future work is the area associated with the modelling of mobile agents. Security is a very important issue when mobility is involved [Jan00]. However, none of the existing agent oriented software engineering methodologies provide concepts and notations to fully capture mobile agents. Although some attempts [Sel03, Kle01, Mou02b, Pog03] have been made to consider the modelling of mobile agents, such attempts are very limited and more work is definitely required in this direction. An interesting direction would be to extend the proposed framework to account for mobile agents, and identify the (more complicated) security issues existing in such systems.

Moreover, although the extensions presented in this thesis are focused to security modelling, some of the concepts introduced, such as the concept of constraint, could be used to model other non-functional requirements. Thus, an interesting direction would be the extension of the proposed approach to allow developers to consider simultaneously many non-functional requirements such as performance and reliability.

Although the pattern language presented in this thesis is complete for the purpose of this research, more patterns can be added in order to extend the applicability of the pattern language. In addition, the SKwyRL framework [Do03] could be used to more precisely define and formalise the patterns according to social, intentional, structural, communicational and dynamic dimensions.

Moreover, an important direction for future work is the development of a tool that will assist developers in the development of a multiagent system by employing the Tropos methodology. Especially with the introduction of security issues the necessity of such a tool is very important. Apart from assisting developers in the development of the system-to-be, this tool could perform more advanced functions such as check automatically the syntax and the consistency of the developed models, automatically produce some attack scenarios and check the system responses to possible attacks, and generate code corresponding to the developed design. A global architecture for such a tool is depicted in Figure 7-1. A developer interacts with the tool through a *Graphical User Interface*. The *Tropos Modeller* component is the main component used for the Tropos models and the modelling activities. It communicates with the *Syntax/Consistency Checker* component, which is responsible for automatically checking the syntax and the consistency of the models created by the developer according to the Tropos syntax specification. Moreover, the *Tropos Modeller* communicates with the *Pattern Repository* to allow developers to use existing patterns during their development (such as the security design patterns proposed by this research), and also with the *Attack Tester* component that automatically generates Security Attack Scenarios and checks how the system copes against possible attacks. Moreover, the *XML Converter* transforms the developed models to eXtensible Markup Language (XML) [Cle01] syntax to allow the generation of implementation code through the *Code Generator* component.

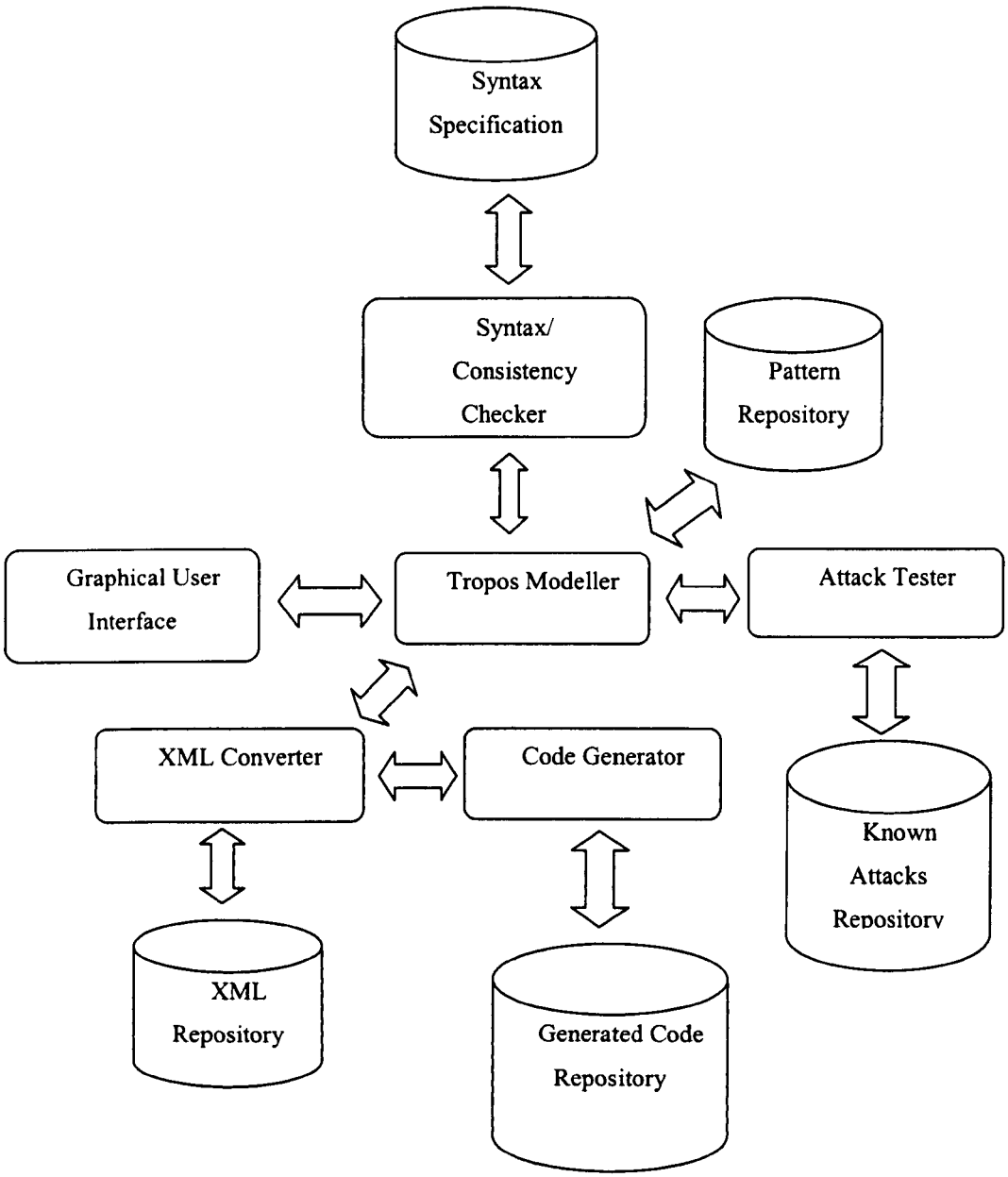


Figure 7-1: A global architecture for a Tropos tool

Future work can also take place from the point of view of the development of the electronic single assessment process. As mentioned above, the presented analysis and design of the electronic single assessment process system is based on the Scenario presented earlier in this thesis. Thus, a complete analysis and design could be produced along with the implementation of the system. Then the system could be tested on a real setting to prove the suitability of agent technology.

7.7 SUMMARY

Although agent oriented software engineering has advanced the last few years, it is still a field in its infancy and many issues needs to be resolved. The integration of security issues in agent oriented software engineering methodologies has been identified as one of the important issues for this paradigm to become widely accepted. Towards this direction, the main aim of this research project was to ***provide an agent oriented software engineering methodology to assist (even non-security oriented) developers in considering security issues during the development of multiagent systems using the same concepts and notations throughout all the development stages.***

This aim has been met by extending the Tropos methodology to enable it to model security issues throughout the development stages using the same concept and notations. The applicability of the approach was tested by applying it to the development of the electronic single assessment process, an agent based system to deliver the single assessment process for older people in England. The application of the proposed approach in the development of the electronic single assessment process indicated that the approach does help developers to successfully consider security issues throughout the development stages.

APPENDIX A: CONSISTENCY RULES

This appendix aims to provide a set of consistency rules discussed in section 5.5. It must be noticed that the presented list of consistency rules cannot be considered an extensive list of all the possible checks that a developer should apply when developing systems with the proposed approach in mind.

The presented list only indicates a set of main rules that should be applied and could help developers to check their design. However, it is more likely that different developers would identify more rules to help them deal with their design and the individual way of thinking and developing a system.

The illustrated rules are divided into two main categories, outer and inner model rules. Outer-model rules describe consistency checks applicable to the security-oriented process as a whole, whereas inner-model rules describe consistency checks that are applicable to individual components of the security-oriented approach.

Consistency rules for the whole process (outer-model)

- All security components must be uniquely labelled.
- Any security components that appear throughout the diagrams must have consistent names across the diagrams.
- If a component appears in a diagram more than once, such duplication should be denoted with an asterisk *.
- Each pattern applied to the development process must be associated to at least one security requirement identified.
- During decomposition, every secure goal of the system must be assigned to at least one agent.
- All the secure goals that the actors delegate responsibility to the system must be satisfied by the system (at least one system internal actor must be assigned to satisfy those goals).

Security reference diagram consistency rules (inner-model)

- Only one security reference diagram is required for each system development.
- A security reference diagram must have at least a security feature and associated protection objectives, security mechanisms and threats.
- Each security feature identified receives only positive contributions from different protection objectives and only negative contributions from the threats. Positive contributions help towards the satisfaction of the security feature while negative contributions put in danger the security feature.
- Each protection objective and each threat that appear on the diagram must be associated with at least one security feature of the graph.
- Each security mechanism that appears on the graph must contribute (either positively or negatively) to at least one protection objective.
- A security mechanism must contribute either positively or negatively to other security mechanisms identified in the graph.
- A protection objective must contribute only negatively to the threats of the security feature it is associated with.

Consistency rules related to security constraints and secure entities modelling (inner model)

- In an actors' diagram, all security constraints must be linked appropriately to at least one dependency.
- If a security constraint is delegated from one actor to another, then the related secure goals must be also delegated.
- During the early requirements analysis, for each security constraint imposed to an actor, a secure goal should be associated to help the actors towards the achievement of the constraint.
- A security constraint modelled in the actors' diagram should appear in the appropriate actor's rationale diagram.

- Security constraint decomposition implies the satisfaction of the root security constraint if and only if all the sub-constraints are satisfied.
- In a rationale diagram, the entities that a security constraint restricts should be clearly marked with a "restricts" link.

Selecting Different Styles diagram (inner model)

- In a selecting styles diagram, each security requirement appear should be traced from the systems rationale diagram.
- In a selecting styles diagram, each link between a style and a requirement should be assigned a weight.
- In a selecting styles diagram, the weights in the links should have a value between 0 and 1.

Security Attack Scenarios (inner model).

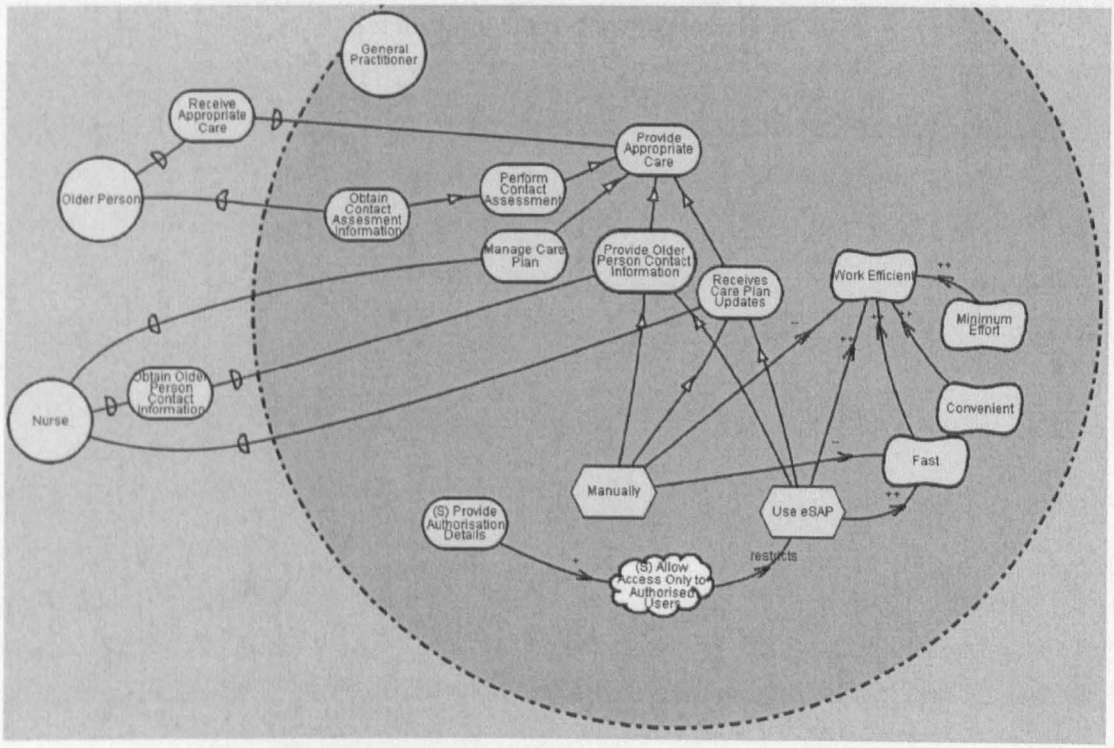
- A name should be defined for each scenario.
- Agents should be represented using the correct notation.
- Attack links and help links should be correctly denoted.
- Only one scenario should exist for the same kind of attack.
- The attack scenarios should include all the agents related to any kind of attack.
- The Prevented and the non-prevented attacks should be correctly marked.

APPENDIX B: SUPPORTED MATERIAL FOR CHAPTER 6

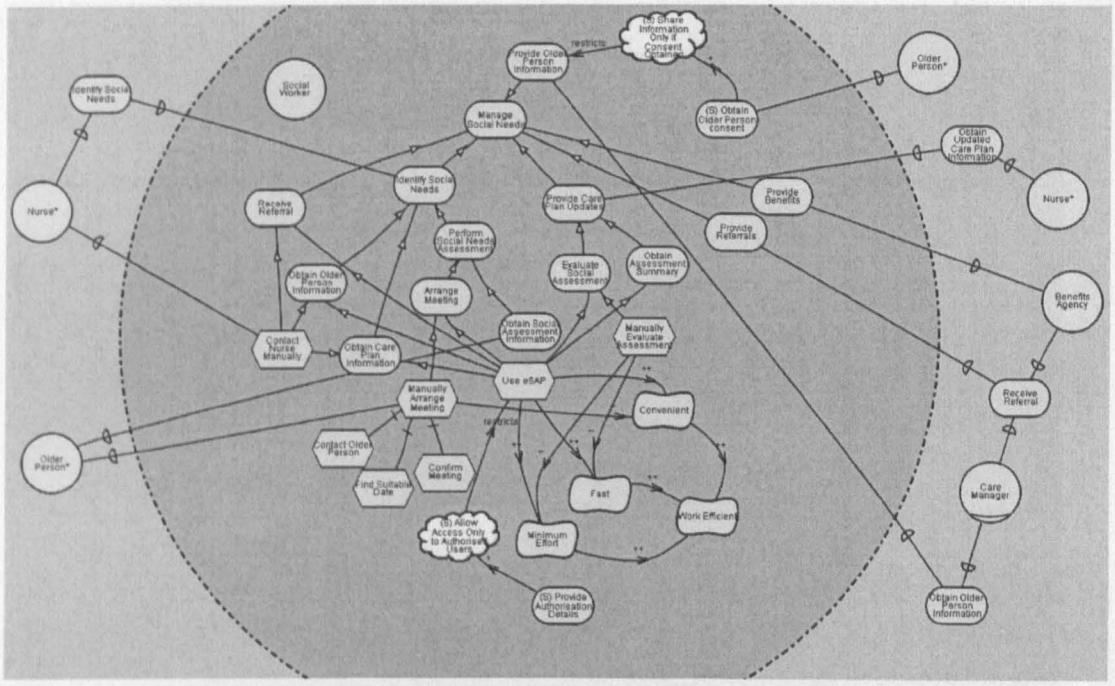
This appendix is divided into three main sections and it provides supported material to the analysis presented in chapter 6. The first section includes goal diagrams that analyse internally all the main actors that were not analysed in chapter 6. The second section presents extended diagrams that indicate the internal actors (those who were not analysed in chapter 6) of the eSAP system and their relationships. The third section provides a list of all the agents of the eSAP system and their capabilities according to the scenario presented in chapter 6.

A) Goal Diagrams

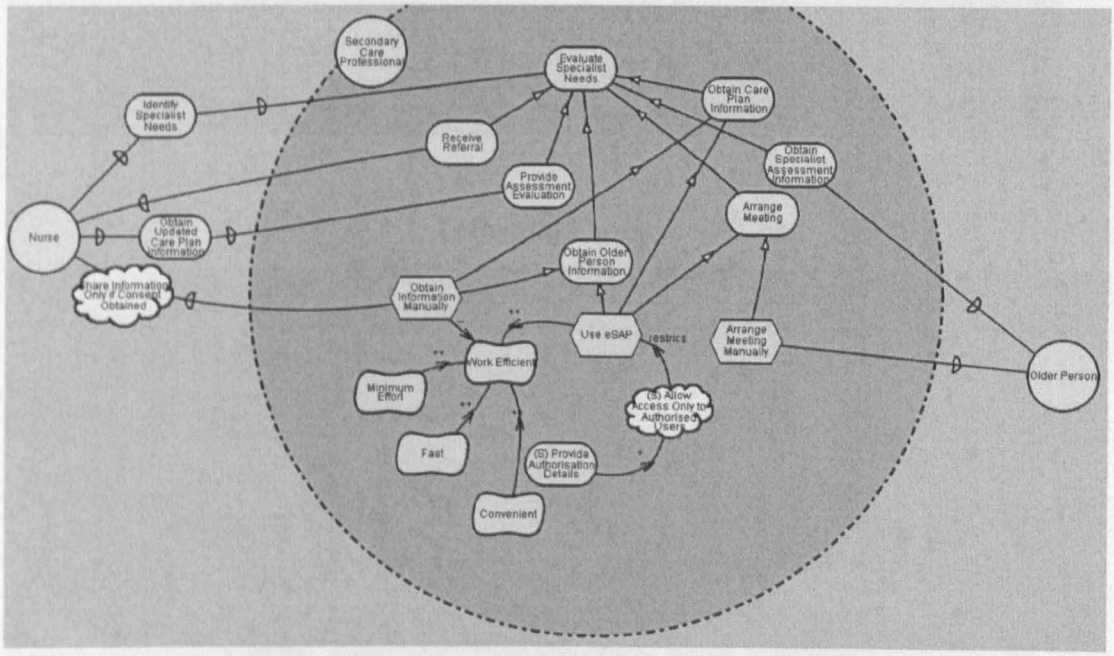
General Practitioner



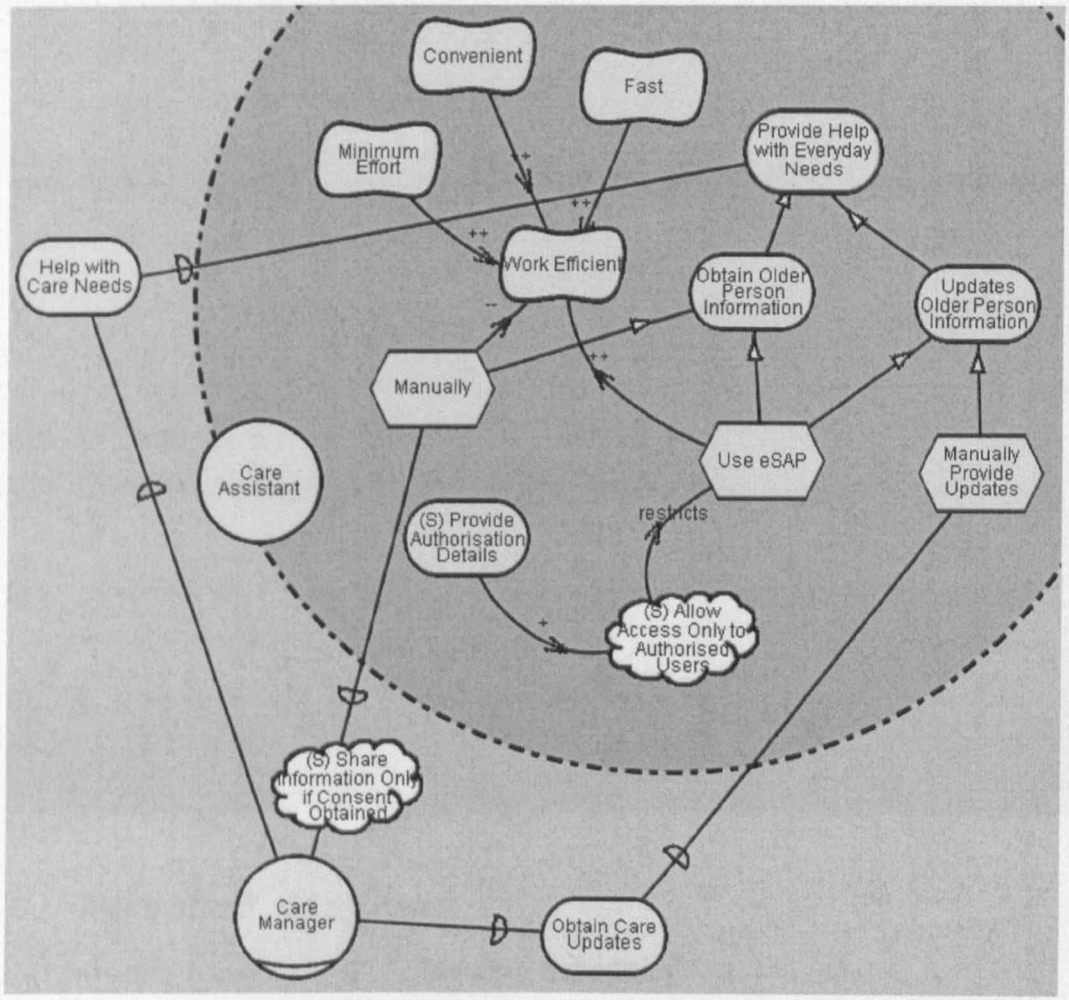
Social Worker



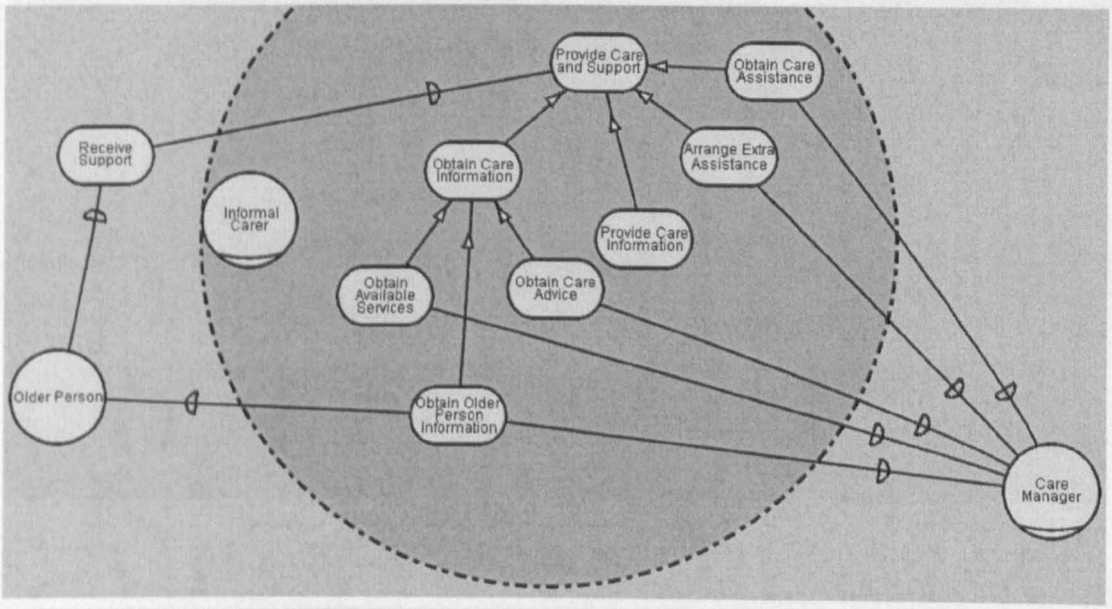
Secondary Care Professional



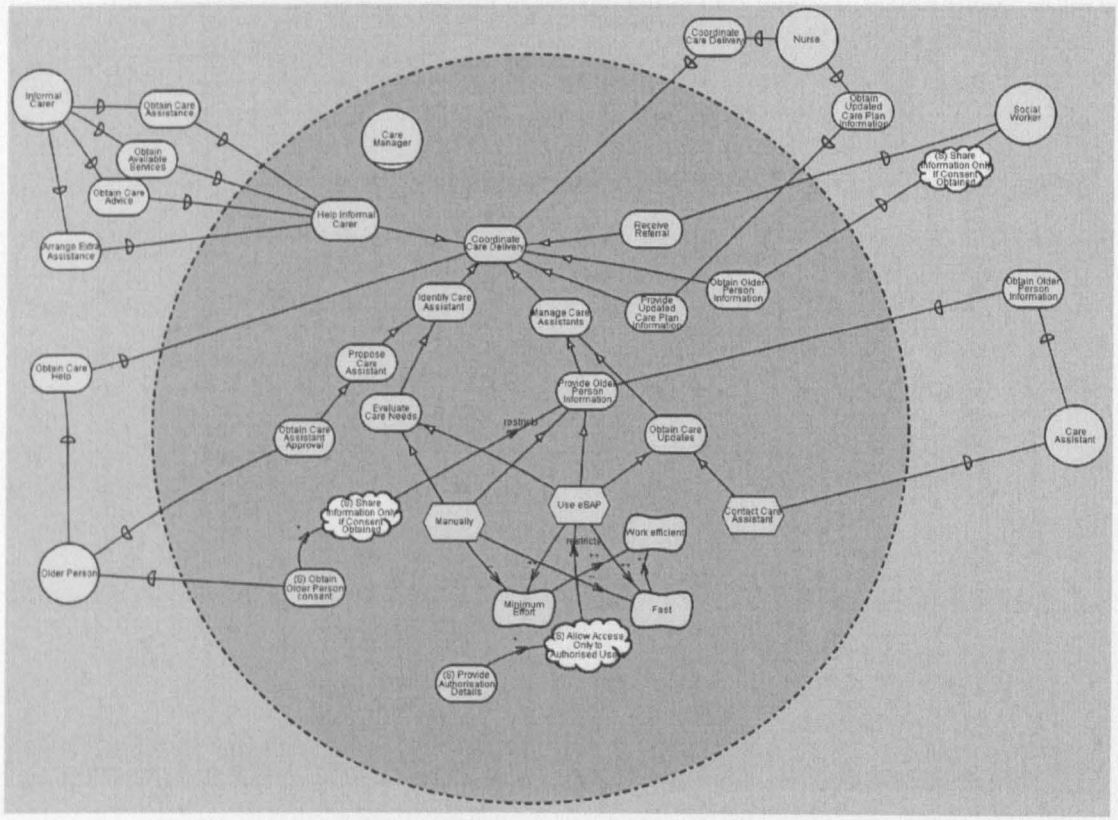
Care Assistant



Informal Carer

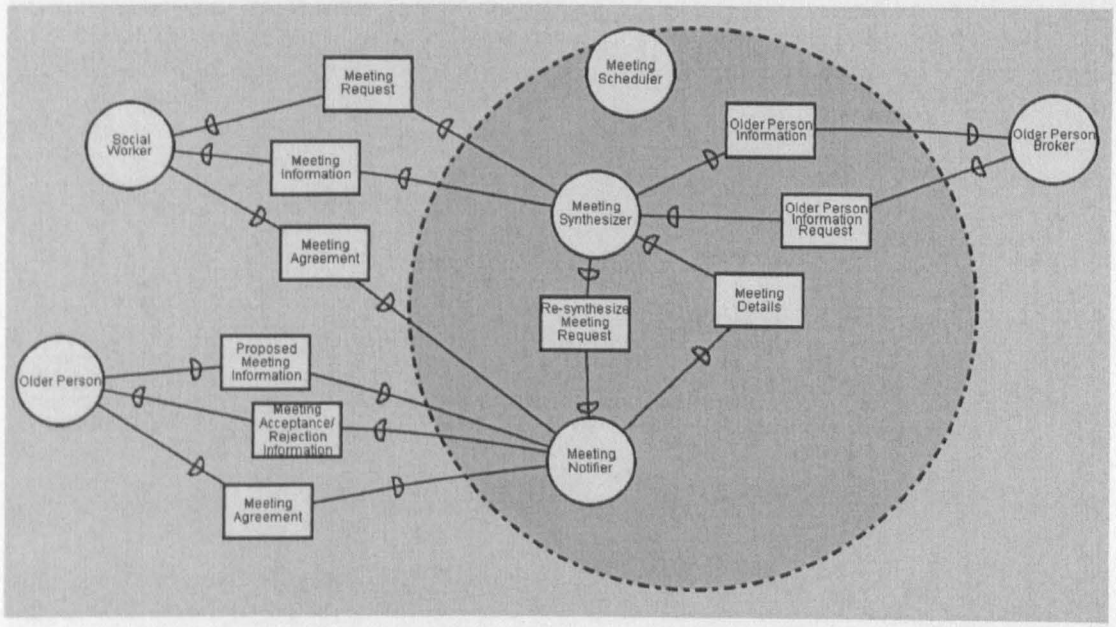


Care Manager

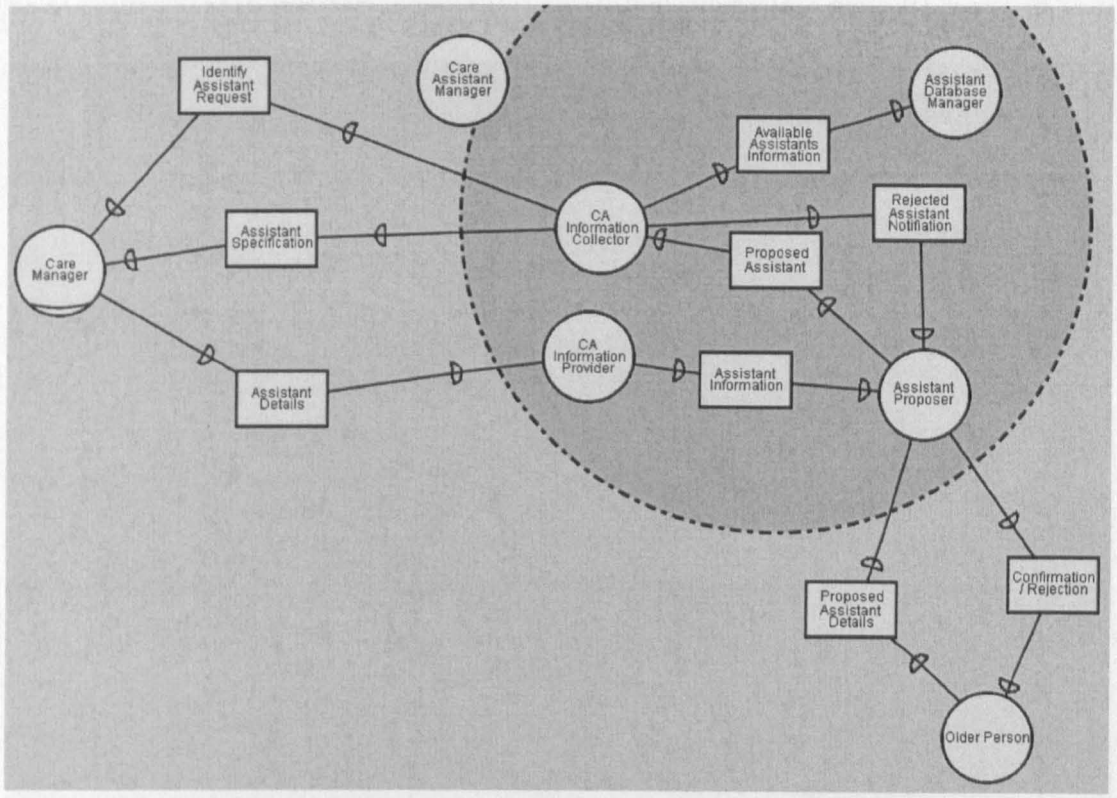


B) Extended diagrams

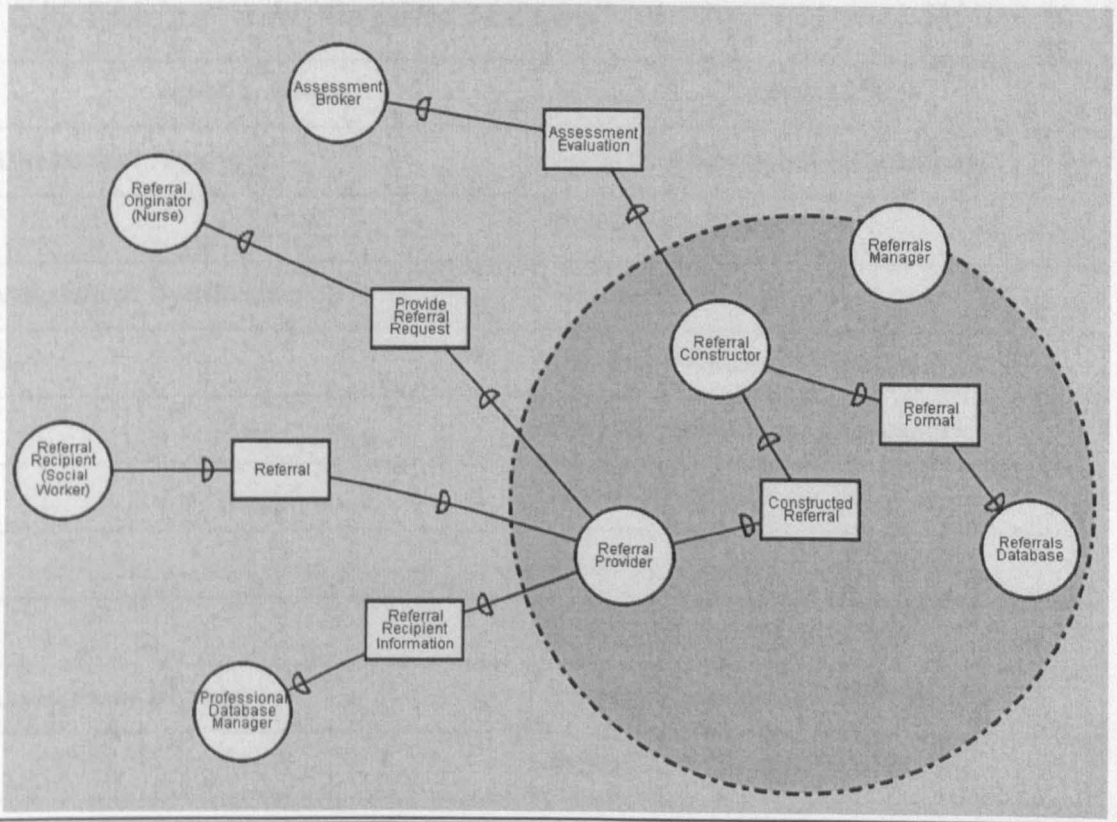
Meeting Scheduler



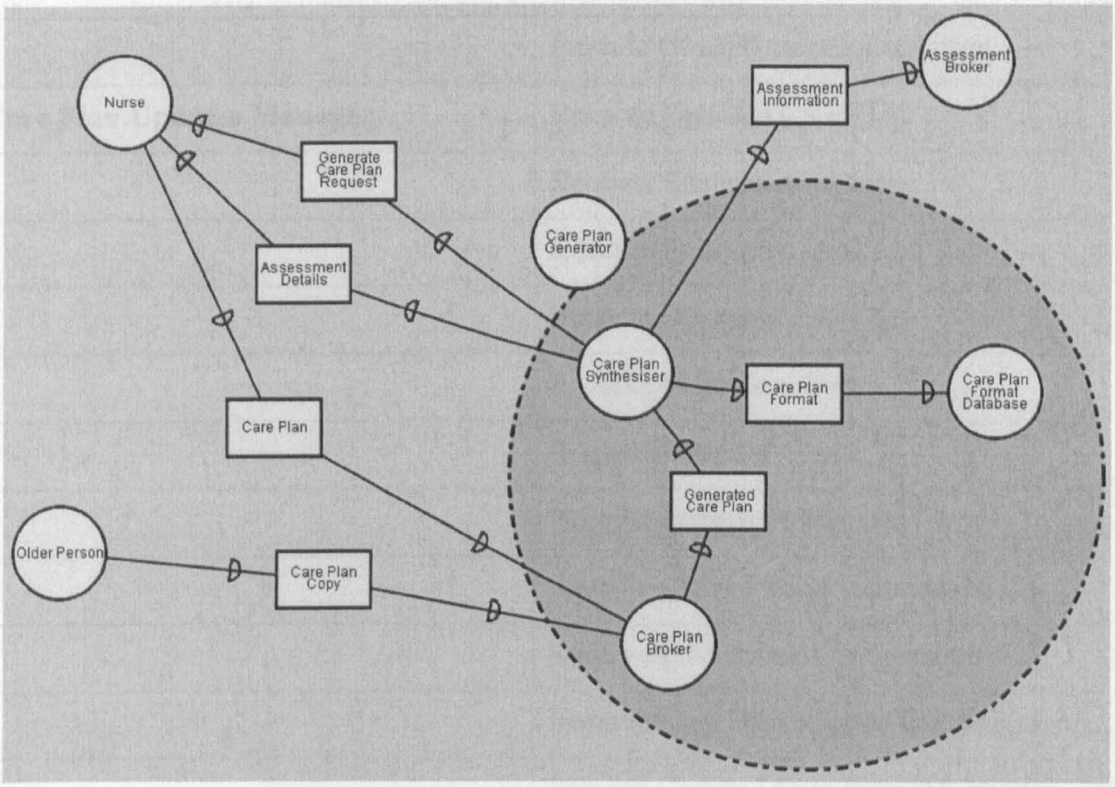
Care Assistant Manager



Referrals Manager



Care Plan Generator



C) Agents and capabilities

Agent Name	Capability
Assessment Analyser	Receive Assessment Information
	Provide Problems
Assessment Synthesizer	Receive Problems
	Receive Evaluation Request
	Provide Assessment Evaluation
	Receive Required Skills
	Receive Available Professionals
	Receive Proposed Actions
Assessment Broker	Provide Assessment Information
	Receive Assessment Evaluation

Older Person Broker	Receive Older Person Information Request
	Provide Older Person Information
Care Plan Updates Manager	Receive Updated Care Plan
	Request Encryption of Data
	Receive Encrypted Data
	Request Decryption of Data
	Receive Plain Text Data
	Request Integrity Check
	Receive Integrity Clearance
	Receive Older Person Information
	Receive Professional Information
	Receive Care Plan Related Information
	Provide Updated Care Plan Information
	Provide Care Plan Information Request
Care Plan Broker	Receive Care Plan Information Request
	Provide Care Plan Related Information
	Request Encryption of Data
	Receive Encrypted Data
	Request Decryption of Data
	Receive Plain Text Data
	Request Integrity Check
	Receive Integrity Clearance
	Receive Updated Care Plan Information
	Receive Care Plan Request
	Provide Care Plan

	Receive Access Control Clearance
	Receive Generated Care Plan
Care Plan Generator	Receive Assessment Information
	Receive Care Plan Request
	Receive Assessment Details
	Provide Care Plan
	Provide Care Plan Copy
Care Plan Format Database	Receive Care Plan Format Request
	Provide Care Plan Format
Referral Provider	Receive Constructed Referral
	Receive Provide Referral Request
	Receive Referral Recipient Information
	Provide Referral
Referral Constructor	Receive Assessment Evaluation
	Receive Referral Format
	Provide Constructed Referral
Referrals Database	Provide Referral Format
Meeting Synthesizer	Receive Meeting Request
	Receive Meeting Information
	Receive Older Person Information
	Provide Older Person Information Request
	Receive Re-synthesize Meeting Request
	Provide Meeting Details
Meeting Notifier	Provide Re-synthesize Meeting Request
	Receive Meeting Details

	Provide Meeting Agreement
	Provide Proposed Meeting Information
	Receive Meeting Acceptance/Rejection Information
CA Information Collector	Receive Identification of Assistant Request
	Receive Assistant Request Specification
	Receive Available Assistants Information
	Receive Rejected Assistant Notification
	Provide Proposed Assistant
CA Information Provider	Provide Assistant Details
	Receive Assistant Information
Assistant Proposer	Provide Assistant Information
	Receive Proposed Assistant
	Provide Rejected Assistant Notification
	Provide Proposed Assistant Details
	Receive Confirmation/Rejection
Consent Manager	Receive Consent Request
	Provide Consent Acceptance/Rejection
Availability Manager	Back up System Files
	Recover System Files
	Provide Back up System Files
Auditing Manager	Monitor System
	Monitor Network
	Provide System Attack Detection
Integrity Verification Manager	Provide Integrity Clearance

	Receive Integrity Request
Cryptography Manager	Receive Encryption Request
	Provide Encrypted Data
	Receive Decryption Request
	Provide Decrypted Data
	Change Cryptographic Algorithm
Access Controller	Receive forwarded Care Plan Request
	Receive Security Policy
	Check Security Policy
	Provide Access Control Clearance
Authenticator	Receive Authentication Request
	Receive Authentication Details
	Provide Authentication Clearance
	Receive Trusted Agencies
eSAP Guard	Receive Access Request
	Provide Authentication Request
	Receive Authentication Clearance
	Provide Access Clearance
Skills Manager	Receive Skills Info Request
	Provide Required Skills
Professional Database Manager	Receive Professional Information Request
	Provide Professional Information
Actions Manager	Receive Actions Request
	Provide Actions Information
Assistants Database Manager	Receive Assistant Information Request

	Provide Available Assistant Information
eSAP Security Policy Manager	Receive Security Policy Request
	Provide Security Policy Information
Trusted Agencies Manager	Receive Trusted Agencies Request
	Provide Trusted Agencies
Filter Agent	Scan eSAP
	Provide Scan Results
Viruses Monitor	Scan eSAP for Viruses
	Provide Scan Results
Social Worker	Provide Assessment Information
	Provide Evaluation Request
	Receive Referral
	Provide Updated Care Plan
	Provide Care Plan Request
	Receive Care Plan
	Provide Consent Request
	Receive Consent Acceptance/Rejection
	Provide System Access Request
	Receive System Access Clearance
	Provide Authorisation Details
	Receive Updated Care Plan Information
	Change Cryptographic Algorithm
	Provide Meeting Request
	Provide Meeting Information
	Receive Meeting Agreement
	Receive Proposed Meeting Information

	Provide Meeting Acceptance/Rejection Information
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity
Nurse	Generate Care Plan Request
	Provide Assessment Details
	Provide Assessment Information
	Provide Evaluation Request
	Receive Assessment Evaluation
	Provide Referral Request
	Receive Referral
	Provide Care Plan Request
	Receive Updated Care Plan Information
	Provide System Access Request
	Receive Care Plan
	Provide Care Plan Request
	Provide Consent Request
	Receive Consent Acceptance/Rejection
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Provide Meeting Request
	Provide Meeting Information
	Receive Meeting Agreement
	Receive Proposed Meeting Information
	Provide Meeting Acceptance/Rejection

	Information
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity
Secondary Care Professional	Provide Specialist Assessment Information
	Provide Evaluation Request
	Receive Referral
	Provide Updated Care Plan
	Provide Care Plan Request
	Receive Care Plan
	Provide System Access Request
	Receive System Access Clearance
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Receive Updated Care Plan Information
	Provide Meeting Request
	Provide Meeting Information
	Receive Meeting Agreement
	Receive Proposed Meeting Information
	Provide Meeting Acceptance/Rejection Information
	Provide Consent Request
	Receive Consent Acceptance/Rejection
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity

General Practitioner	Provide Older Person Contact Information
	Receive Care Plan Updates
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Receive Contact Assessment Information
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity
	Provide System Access Request
	Receive System Access Clearance
Older Person	Receive Meeting Request
	Accept/Reject Meeting Request
	Receive Updated Care Plan Information
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Provided Contact Assessment Information
	Provide Overview Assessment Information
	Provide Specialist Assessment Information
	Provide Social Assessment Information
	Receive Care Plan Copy
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity

	Receive Consent Request
	Provide/Reject Consent
Care Assistant	Obtain Older Person Information
	Provide Older Person Information Updates
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity
Care Manager	Provide Assistant Approval Request
	Receive Care Assistant Approval
	Provide Assistant Specification
	Receive Assistant Details
	Provide Older Person Consent Request
	Receive Consent Acceptance/Rejection
	Receive Care Plan Updates
	Receive Older Person Information
	Provide Authorisation Details
	Change Cryptographic Algorithm
	Encrypt Transmitted Data
	Decrypt Received Data
	Check Data Integrity

REFERENCES

[Ale79] C. Alexander, *The Timeless way of Building*, Volume 1, Oxford University Press, New York, 1979.

[Amo94] E. Amoroso, *Fundamentals of Computer Security Technology*, Prentice-Hall, 1994.

[And01] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley Computer Publishing, 2001.

[Andr99] M. Andries, G. Engels, A. Habel, B. Hoffmann, H-J Kreowski, S. Kuske, D. Plump, A. Schurr, G. Taentzer, *Graph Transformation for Specification and Programming*, Science of Computer Programming, 1999.

[Ant94] A. I. Anton, W.M. McCracken, C. Potts, Goal Decomposition and Scenario Analysis in Business Process Reengineering, In *Proceedings of the 6th Conference on Advanced Information Systems* (CAiSE-1994), Utrecht-The Netherlands, 1994.

[Ban89] M. Burrows, M. Abadi, R. Needham, A logic of authentication, In *Proceedings of the Royal Society of London* A, 426, pp 233-271, 1989.

[Bas98] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, *SEI Series in Software Engineering*, Addison – Wesley, 1998.

[Bau01] B. Bauer, J. Müller, J. Odell, Agent UML: A Formalism for Specifying Multiagent Interaction, *In Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge (eds.), Lecture Notes in Computer Science, pp. 91-103, Springer, Berlin, 2001.

[Bec94] K. Beck, R. Johnson, Patterns Generate Architectures, *In Object Oriented Programming: 8th European Conference on Object-Oriented Programming* (ECOOP 1994), M. Tokoro and R. Pareschi (Eds.), Lecture Notes in Computer Science 821, pp.139-149, Springer, Berlin, 1994.

[Ber98] P. Bertrand, R. Darimont, E. Delor, P. Massonet, A. Van Lamsweerde. GRAIL/KAOS: an environment for goal driven requirements engineering, *In Proceedings of the 20th International Conference on Software Engineering* (ICSE'98), IEEE-ACM, Kyoto, April 98

[Bey95] Paul Beynon-Davies, Information systems `failure': case of the LASCAD project, *European Journal of Information Systems*, 1995

[Bir86] N. D. Birrell, M.A. Ould, *A practical handbook for software development*, Campridge University Press, 1986

[Boe81] B. W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[Boe84] B. W. Boehm, Verifying and Validating Software Requirements and Design Specifications, *IEEE Software*, Vol.1, No. 1, January 1984.

[Boo94] G. Booch, *Object-oriented analysis and design – with applications*, The Benjamin / Cummings Publishing Company, 1994.

[Bra97] M. Bradshaw, *Software Agents*, American Association Artificial Intelligence Publication, 1997.

[Bre02] P. Bresciani, P. Giorgini, The Tropos Analysis Process as Graph Transformation System, *In Proceedings of the Workshop on Agent-oriented methodologies*, at OOPSLA 2002, Seattle, WA, USA, Nov, 2002.

[Bre02a] P. Bresciani, A. Perini, P. Giorgini, G. Giunchiglia, J. Mylopoulos, Modelling early requirements in Tropos: a transformation based approach, In *Agent Oriented Software Engineering II.* M. Wooldridge, and G. Weiβ (eds.). Lecture Notes in Computer Science, Springer-Verlag 2222, 2002.

[Bre02b] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos A. Perini, TROPOS: An Agent Oriented Software Development Methodology, Submitted to the *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers.

[Bre03] P. Bresciani, F. Sannicolò, Requirement analysis in TROPOS: a self referencing example, In *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, R. Kowalszyk, J. Müller, H. Tianfield, and R. Unland, (eds.), Lecture Notes in Artificial Intelligence, Springer-Verlag 2592, 2003.

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Soomerlad, M. Stal, *Pattern Oriented Software Architecture: A System of Patterns*, Willey, 1996.

[Car91] J. M. Carroll, M. B. Rosson, Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenario, *IBM Research Report*, Human Computer Interaction, RC 17908 (75365), 1991.

[Cas01] J. Castro, M. Kolp, J. Mylopoulos, A Requirements-Driven Development Methodology, In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering* (CAiSE'01), pp. 108-123, Interlaken- Switzerland, 2001.

[Cas02] J. Castro, M. Kolp, J. Mylopoulos, Towards Requirements-Driven Information Systems Engineering: The Tropos project, In *Information Systems* (27), pp 365-389, Elsevier, Amsterdam - The Netherlands, 2002.

[Chu95] L. Chung, B. Nixon, Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach, In *Proceedings of the 17th International Conference on Software Engineering*, Seattle- USA, 1995.

[Cle01] J. G. Cleaveland, *Program Generator with XML and Java*, Prentice-Hall, 2001

[Col94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes, *Object Oriented Development: The FUSION Method.* Prentice Hall International, 1994

[Dar91] Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed Requirements Acquisition, Science of Computer Programming, Special issue on the *6th International workshop of Software Specification and Design*, 1991.

[Deu01] D. Deugo, M. Weiss, E. Kendall, Reusable Patterns for Agent Coordination, In *Coordination of Internet Agents*, Springer-Verlag, 2001

[Dev00] P. Devanbu, S. Stubblebine, Software Engineering for Security: a Roadmap, In *Proceedings of the conference of The future of Software engineering*, 2000.

[Do03] T. T. Do, M. Kolp, T. T. Hang Hoang, A. Pirotte, A Framework for Design Patterns for Tropos, *In Proceedings of the 17th Brazilian Symposium on Software Engineering (SBES 2003)*, Maunas, Brazil, October 2003.

[Doh03] Department of Health, *Single Assessment Process for Older People*, http://www.doh.gov.uk/scg/sap/, last accessed 16/12/2003

[Eco99] The Economist, *Digital rights and wrongs*, July 17, 1999

[Eri98] H. E. Eriksson, M Penker, *Unified Modelling Language (UML) toolkit*, Wiley Computer Publishing, 1998

[Eva01] R. Evans, P. Kearney, J. Stark, G. Caire, F. J. Carijo, J. J. Gomez Sanz, J. Pavon, F. Leal, P. Chainho, and P. Massonet. MESSAGE: Methodology for Engineering Systems of Software Agents, *AgentLink Publication*, September 2001

[Fai85] R. Fairley, *Software Engineering Concepts*, New York: McGraw-Hill, 1985

[Fer99] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison-Wesley Publishing Company, 1999

[Fern01] E. B. Fernandez, R. Pan, A Pattern Language for Security Model, *Proceedings of the 8th Conference on Pattern Languages of Programs* (PLoP 2001), Monticello, Illinois, USA, September 2001

[Fern02] E. B. Fernandez, Patterns for Operating Systems Access Control, *Proceedings of the 9th Conference on Pattern Languages of Programs* (PLoP 2002), Monticello, Illinois, USA, September 2002.

[Fic92] R. G. Fichman, C. F. Kemerer, Object-Oriented and Conventional Analysis and Design Methodologies, *IEEE Computer*, Vol. 25, No 10, pp 22-39, Oct 1992

[Fis02] K. Fischer, D. Hutter, M. Klusch, W. Stephan, Towards Secure Mobile Multiagent Based Electronic Marketplace Systems, *Electronic Notes in Theoretical Computer Science*, Vol. 63, Elsevier, 2002.

[Fow97] M Fowler, *Analysis Patterns: Reusable Object Models*, Object Technology Series, Addison-Wesley Publishing Company, Reading, 1997.

[Fow00] M. Fowler, K. Scott, *UML Distilled: A brief guide to the standard Object Modelling Language* (2nd Edition), Addison-Wesley, 2000.

[Fra96] S. Franklin, A. Graesser, Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents, In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[Fux01] A. Fuxman, *Formal Analysis of Early Requirements Specifications,* MSc Thesis, University of Toronto, Canada, 2001.

[Fux03] A. Fuxman, L. Liu, M. Pistore, M. Roveri, J. Mylopoulos, Specifying and Analyzing Early Requirements: Some Experimental Results. *In Proceedings of the 11th IEEE International Requirements Engineering Conference*, 8th-12th September 2003, Monterey Bay, California U.S.A

[Gam95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[Gio02] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani. Reasoning with Goal Models, In the *Proceedings of the 21st International Conference on Conceptual Modeling* (ER2002), Tampere, Finland, October 2002.

[Gio03] P. Giorgini, F. Massacci, J. Mylopoulos, Requirement Engineering meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard, *In Proceedings of the 22nd International Conference on Conceptual Modeling* (ER'03), Chicago, Illinois, 13-16 October, 2003.

[Giu02] F. Giunchiglia, J. Mylopoulos, A. Perini, The Tropos Software Development Methodology: Processes, Models and Diagrams, *Lecture Notes in Computer Science 2585*, pp 162-173, Springer 2003

[Gol01] D. Gollmann, *Computer Security*, John Willey and Sons, July 2001

[Gra02] Sarah Granger, *Social Engineering Fundamentals*, Part I: Hacker Tactics, WWDocument http://online.securityfocus.com/infocus/1527, Last Accessed: 30/9/2002

[Hew77] C. Hewitt, Viewing Control Structures as Patterns of Message Passing, *Artificial Intelligence*, Vol. 8, No 3, pp 323-374, 1977

[Hub97] H. Hubmann, Formal Foundations for Software Engineering Methods, *Lecture Notes in Computer Science 1322*, Springer- Verlag, 1997

[Hug02] M.-P. Huget, Nemo: An Agent-Oriented Software Engineering Methodology, *In Proceedings of OOPSLA Workshop on Agent-Oriented Methodologies*, John Debenham, Brian Henderson-Sellers, Nicholas Jennings and James Odell (eds), Seattle, USA, November 2002.

[IEEE90] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 729, 1990.

[Igl96] Carlos A. Iglesias, Mercedes Garijo, José C. González, Juan R. Velasco, A Methodological Proposal for Multiagent Systems Development extending CommonKADS, *In Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 25-1/17, Banff, Canada, 1996.

[Igl97] C. A. Iglesias, M. Garijo, J. Gonzalez, J. R. Velasco, Analysis and design of multiagent systems using MAS-CommonKADS, *Workshop on Agent Theories, Architectures and Languages*, 1997.

[Igl97a] Carlos A. Iglesias, Mercedes Garijo, José C. González, Juan R. Velasco, MAS-CommonKADS: A comprehensive Agent-Oriented Methodology, *Proceedings of the 11th Int. Conference on Mathematical and Computer Modelling and Scientific Computing*, Washington, USA, 1997

[Igl99] C. Iglesias, M. Garijo, J. Gonzales, A survey of agent-oriented methodologies, *Intelligent Agents IV*, Lecture Notes in Computer Science, Springer-Verlag 1555, 1999.

[Jac99] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley Publishing Company, 1999.

[Jack90] P. Jackson, *Introduction to Expert Systems*, Addison-Wesley Inc. 1990

[Jan00] W. Jansen, Countermeasures for Mobile Agent Security, Computer Communications, *Special Issue on Advanced Security Techniques for Network Protection*, Elsevier Science BV, November 2000.

[Jan99] W. Jansen, T. Karygiannis, Mobile Agent Security, *National Institute of Standards and Technology*, Special Publication 800-19, August 1999.

[Jaw00] J. Jaworski, P. J. Perrone, *Java Security Handbook*, SAMS, Indianapolis, IN, 2000

[Jen01] N. R. Jennings, An agent-based approach for building complex software systems, *Communications of the ACM*, Vol. 44, No 4, April 2001

[Jen99] N. R. Jennings, M. Wooldridge, Agent–Oriented Software Engineering, *in the Proceedings of the 9th European Workshop on Modelling Autonomous Agents in*

a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99), Valencia, Spain, 1999

[Jur01] Jan Jürjens, Towards Secure Systems Development with UMLsec, *Fundamental Approaches to Software Engineering* (FASE/ETAPS) 2001, International Conference, Genoa 4-6 April 2001

[Jur02] J. Jürjens, UMLsec: Extending UML for Secure Systems Development, UML 2002, *Lecture Notes in Computer Science 2460*, pp 412-425, Springer 2002

[Kaz94] R. Kazman, G. Abowd, L. Bass, M. Webb, SAAM: A Method for Analyzing the Properties of Software Architectures, *Proceedings of ICSE-16*, Sorrento –Italy, May, 1994

[Kin96] D. Kinny, M. Georgeff, A. Rao, A Methodology and Modelling Technique for Systems of BDI Agents, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Lecture Notes in AI, Springer- Verlag, Vol. 1038, 1996

[Kle01] C. Klein, A. Rausch, M. Sihling, Z. Wen, Extension of the Unified Modeling Language for Mobile Agents, *In Unified Modeling Language: Systems Analysis, Design and Development Issues*, edited by Keng Siau and Terry Halpin, Idea Group Publishing Book, 2001

[Kol01] M. Kolp, P. Giorgini, J. Mylopoulos, A Goal-Based Organizational Perspective on Multi-Agent Architectures, *in the Proceedings of the 8th International Workshop on Agent Theories, architectures, and languages* (ATAL-2001), Seattle-USA, August 2001.

[Kos97] G. Kosters, B.U. Pagel, M. Winter, Coupling Use Cases and Class Models, *Proceedings of the BCS-FACS/EROS workshop on "Making Object Oriented Methods More Rigorous"*, Imperial College, London-England, 1997.

[Lal95] V. Lalioti, C. Theodoulidis, Visual Scenarios for Validation of Requirements Specification, *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering* (SEKE'95), Rochville, Maryland-USA, 1995.

[Lam00] A. van Lamsweerde, Requirements engineering in the year 00: a research perspective, in the *Proceedings of the 22nd International Conference on Software Engineering* (ICSE 2000), pp. 5-19, ACM, 2000.

[Lan85] V. P. Lane, *Security of Computer Based Information Systems*, Macmillan education ltd, 1985

[Lang99] D. B. Lange, M. Oshima, Seven Good Reasons for Mobile Agents, *Communications of the ACM*, Vol. 42, No 3, March 1999.

[Leh02] S. Lehtonen, J. Pärssinen, A Pattern Language for Cryptographic Key Management, *Proceedings of the 7th European Conference on Pattern Languages of Programs* (EuroPLoP), Irsee, Germany, June 2002.

[Lin01] J. Lind, Iterative software engineering for multiagent systems: the MASSIVE method, *Lecture Notes in Computer Science*, Springer-Verlag, 2001.

[Liu02] L. Liu, E. Yu, J. Mylopoulos, Analyzing Security Requirements as Relationships Among Strategic Actors, in the *Proceedings of the 2nd Symposium on Requirements Engineering for Information Security* (SREIS'02), Raleigh, North Carolina, October 2002.

[Lod02] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-Based Modelling Language for Model-Driven Security, in the *Proceedings of the 5th International Conference on the Unified Modeling Language*, 2002.

[Mac90] A. Macro, J. Buxton, The craft of software engineering, *International Computer Science Series*, Addison-Wesley Publishing Company, 1990.

[Mae95] P. Maes, Artificial Life Meets Entertainment: Life like Autonomous Agents, *Communications of the ACM*, 38, 11,pp. 108-114, 1995

[Mcd99] J. McDermott, C. Fox, Using Abuse Care Models for Security Requirements Analysis, *Proceedings of the 15th Annual Computer Security Applications Conference*, December 1999.

[Mea94] C. Meadows, A Model of Computation for the NRL protocol analyser, *Proceedings of the 1994 Computer Security Foundations Workshop*, 1994.

[Mou02] H. Mouratidis, P. Giorgini, G. Manson, I. Philp, A Natural Extension of Tropos Methodology for Modelling Security, In the *Proceedings of the Agent Oriented Methodologies Workshop* (OOPSLA 2002), Seattle-USA, November 2002.

[Mou02a] H. Mouratidis, P. Giorgini, I. Philp, G. Manson, Using Tropos Methodology to Model and integrated Health Assessment System, In *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information systems* (AOIS-02) at CAiSE2002, Toronto, Canada, 2002.

[Mou02b] H. Mouratidis, J. Odell, G. Manson, Extending the Unified Modeling Language to Model Mobile Agents, in the *Proceedings of the Agent Oriented Methodologies Workshop* (at the OOPSLA 2002), Seattle - USA, November 2002.

[Mou03] H. Mouratidis, P. Giorgini, G. Manson, Modelling Secure Multiagent Systems, in the *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne-Australia, pp. 859-866, ACM 2003.

[Mou03a] H. Mouratidis, P. Giorgini, M. Schumacher, G. Manson, Security Patterns for Agent Systems, *Proceedings of the Eight European Conference on Pattern Languages of Programs* (EuroPLoP), Irsee, Germany, June 2003.

[Mou03b] H. Mouratidis, P. Giorgini, M. Weiss. Integrating Patterns and Agent-Oriented Methodologies to Provide Better Solutions for the Development of Secure Agent-Based Systems, *Proceedings of the 6th ChiliPLoP Annual Conference, Hot Topic: Expressiveness of Pattern Languages*, Arizona, USA, March 2003.

[Mou03c] H. Mouratidis, I. Philp, G. Manson, A Novel Agent-Based System to Support the Single Assessment Process for Older People, in the *Journal of Health Informatics* (9) 3, pp. 149-163, September 2003.

[Mou03d] H. Mouratidis, P. Giorgini, G. Manson, An Ontology for Modelling Security: The Tropos Approach, in the *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems* (KES 2003), Invited Session on Ontology and Multi-Agent Systems Design (OMASD'03), Oxford-England, September 2003.

[Nau68] P. Naur, B. Randell, Software Engineering – Report on a conference, Garmisch, *NATO Scientific Affairs Division*, 1968.

[Nor96] Norman L. Kert, John M. Vlissides, James O. Coplien, *Pattern Languages of Program Design 2*, Addison Wesley Publishing, 1996.

[Nus01] B. Nuseibeh, S. Easterbrook, A. Russo, Making Inconsistency Respectable in Software Development, *Journal of Systems and Software*, 58(2):171-180, Elsevier Science Publishers, September, 2001.

[Nwa96] H.S. Nwana, Software Agents: An Overview, *Knowledge Engineering Review*, Vol. 11, No 3, pp 205-244, 1996

[Ode99] J. Odell, C. Bock, Suggested UML extensions for agents, *Technical report, OMG*, December 1999. Submitted to the OMG's Analysis and Design Task Force in response to the Request for Information enti-tled "UML2.0 RFI".

[Per01] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Towards an Agent Oriented approach to Software Engineering, *Proceedings of the Workshop Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena - Italy, 4-5 Sept 2001.

[Phi97] I. Philp, Can a medical and social assessment be combined? *Journal of the Royal Society of Medicine*, 90(32), pp 11-13,1997.

[Plu02] D. Plump, Essentials of Term Graph Rewriting, *Electronic Notes in theoretical Computer Science 51*, 2002.

[Pog03] A. Poggi, G. Rimassa, P. Turci, J. Odell, H. Mouratidis, G. Manson, Modeling Deployment and Mobility Issues in Multiagent Systems using AUML, in the *Proceedings of the 4th International Workshop on Agent Oriented Software Engineering* (AOSE-2003), Melbourne- Australia, 2003.

[Pot94] C. Potts, K. Takahashi, A.I. Anton, Inquiry Based Requirements Analysis, *IEEE Software*, March 1994.

[Roh02] S. Rohrig, Using Process Models to Analyze Health Care Security Requirements, *International Conference Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, L'Aquila, Italy, January, 2002.

[Rom85] G.C. Roman, A Taxonomy of Current Issues in Requirements Engineering, *IEEE Computer*, Vol. 18, No. 4, pp 14-23, April 1985.

[Rum91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorersen, *Object Oriented Modelling and Design*, Prentice Hall, 1991.

[Rus00] D. J. Russel, *FAD: A Functional Analysis and Design methodology*, PhD Thesis, The University of Kent at Canterbury, August 2000.

[Rya00] P. Ryan, S. Schneider, *Analysis and Design of Security Protocols*, Pearson Professional Education, 2000.

[Rys00] J. Ryser, M. Glinz, SCENT - A Method Employing Scenarios to Systematically Derive Test Cases for System Test, *Technical Report 2000.03*, Institut für Informatik, University of Zurich, 2000

[Rys99] J. Ryser, M. Glinz, A Practical Approach to Validating and Testing Software Systems Using Scenarios, *Proceedings of the Third International Software Quality Week Europe* (QWE'99), Brussels, Belgium, November, 1999.

[Sal75] J. Saltzer, M.D.Schroeder, The Protection of information in computer systems, In the *Proceedings of the IEEE* 63 (9), pp.1278-1308, September 1975.

[Sch00] B. Schneier, *Secrets & Lies: Digital Security in a Networked World*, John Wiley & Sons, 2000.

[Schr99] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt, W. Van de Velde, B. Wielinga, Knowledge Engineering and Management, The CommonKADS Methodology, The *MIT press*, December 1999

[Schu01] M. Schumacher, U. Roedig, Security Engineering with Patterns, in the *Proceedings of the 8th Conference on Pattern Languages for Programs* (PLoP 2001), Illinois-USA, September 2001.

[Sco01] Scott A. DeLoach, Mark F. Wood, Clint H. Sparkman, Multiagent Systems Engineering, *The International Journal of Software Engineering and Knowledge Engineering*, Volume 11 no. 3, June 2001

[Sco02] Scott A. DeLoach, Modeling Organizational Rules in the Multiagent Systems Engineering Methodology, *Proceedings of the 15th Canadian Conference on Artificial Intelligence* (AI'2002), Calgary, Alberta, Canada. May 27-29, 2002.

[Sel03] Athie Self, Scott A. DeLoach, Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology, Special Track on Agents,

Interactions, Mobility, and Systems (AIMS) at the *18th ACM Symposium on Applied Computing* (SAC 2003), Melbourne, Florida, USA, March 9 - 12, 2003.

[Sha98] C. Shakeri, *Discovery of Design Methodologies for the Integration of Multi-Disciplinary Design Problems*, PhD thesis, Worchester Polytechnic Institute, 1998.

[Sin00] G. Sindre, A. L. Opdahl, Eliciting Security Requirements by Misuse Cases, *Proceedings of TOOLS Pacific 2000*, November 2000.

[Som01] I. Sommerville, *Software Engineering* – Sixth Edition, Addison-Wesley Publishing Company, 2001.

[Som99] I. Sommerville, P. Sawyer, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons Ltd, October 1999.

[Sta99] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Second Edition, Prentice-Hall 1999.

[Ste95] E. Steegmans, J. Lewi, M. D'Haese, J. Dockx, D. Jehoul, B. Swennen, S. Van Baelen, P. Van Hirtum, EROOS Reference Manual Version 1.0, Department of Computer Science, K.U.Leuven, *CW Report 208*,176 p. Leuven, B, 1995.

[Stu03] A. Sturm, O. Shehory, A Framework for Evaluating Agent-Oriented Methodologies, *Workshop on Agent-Oriented Information Systems* (AOIS), Melbourne, Australia, July 14, 2003.

[Try97] T. Tryfonas, E. Kiountouzis, A. Poulymenakou, Embedding security practices in contemporary information systems development approaches, *Information Management & Computer Security*, Vol 9 Issue 4,pp 183-197, 2001.

[Vli93] H. Van Vliet, *Software Engineering- Principles and Practice*, John Willey and Sons Publishing, 1993.

[Wood01] Mark Wood, Scott A. DeLoach, An Overview of the Multiagent Systems Engineering Methodology, in *Agent-Oriented Software Engineering*, P. Ciancarini, M. Wooldridge, (Eds.), Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001.

[Wool00] M. Wooldridge, N. R. Jennings, D. Kinny, The GAIA methodology for agent-oriented analysis and design, *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No 3, pp 285-312, 2000.

[Wool01] M. Wooldridge, P. Ciancarini, Agent-Oriented Software Engineering: The State of the Art, In P. Ciancarini and M. Wooldridge (eds.), Agent-Oriented Software Engineering. Springer-Verlag, *Lecture Notes in AI Volume 1957*, January 2001.

[Wool95] M. Wooldridge, N. R. Jennings, Agent Theories, Architectures, and Languages: A Survey, *Intelligent Agents*, Wooldridge, Jennings (eds.), Springer-Verlag, pp 1-22, 1995.

[Wool99] M. Wooldridge, N. R. Jennings, D. Kinny, A Methodology for Agent-Oriented Analysis and Design. In O. Etzioni, J. P. Muller, and J. Bradshaw (eds.), *Agents '99: Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, May 1998.

[Yod97] J. Yoder, J. Barcalow, Architectural Patterns for Enabling Application Security, *Proceedings of the 4th Conference on Pattern Languages of Programs* (PLoP 1997), Monticello, Illinois, USA, September 1997.

[Yu01] E. Yu, Agent-Oriented Modelling: Software Versus the World, in the *Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering*, Lecture Notes in Computer Science 2222, pp 206-225, Springer Verlag, 2001.

[Yu01a]E. Yu, Agent Orientation as a Modelling Paradigm, *Wirtschaftsinformatik*, 43 (2), pp 123-132 April 2001.

[Yu02] E. Yu, L. Cysneiros, Designing for Privacy and Other Competing Requirements, *2nd Symposium on Requirements Engineering for Information Security* (SREIS' 02), Raleigh, North Carolina, 15-16 November, 2002.

[Yu95] E. Yu, *Modelling Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[Yu97] E. Yu, Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, *Proceedings of the 3rd IEEE Int. Symposium on*

Requirements Engineering, pp. 226-235, Jan. 6-8, 1997, Washington D.C.-USA, 1997.

[Zam01] F. Zambonelli, N. R. Jennings, M. Wooldridge, Organisational Abstractions for the Analysis and Design of Multi-Agent Systems, P. Ciancarini and M. Wooldridge (eds.), Agent-Oriented Software Engineering, Springer-Verlag, *Lecture Notes in AI*, Vol. 1957, January 2001.