Abstract. This deliverable presents the results of the WP2-Task 2.4 and notably the study and definition of multi-agent patterns. Once defined the organizational structure, the architectural design consists of a phase of micro level design in which the agents are introduced and each agent is responsible for its goals and it can interact with the other agents in order to achieve its goals. We present here a number of catalogue of multi-agent patterns where each pattern is a solution for achieving a particular goal of some agents. In particular, we have considered security and privacy patterns, multi-agent secure interactions and multi-agent databases.
Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’Istruzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A..
Executive Summary

This deliverable presents the results of the WP2-Task 2.4 and notably the study and definition of multi-agent patterns. Once defined the organizational structure, the architectural design consists of a phase of micro level design in which the agents are introduced and each agent is responsible for its goals and it can interact with the other agents in order to achieve its goals. We present here a number of catalogue of multi-agent patterns where each pattern is a solution for achieving a particular goal of some agents. We have considered security and privacy patterns, multi-agent secure interactions and multi-agent databases.

In particular, this deliverable firstly concentrates on modelling secure systems by using agent-oriented approach and security patterns. Secondly, it provides two case studies on using security requirements engineering methodology in practice. Then, it discusses a logical framework for reasoning about access control for autonomic communication. Finally, it overviews state of the art in peer-to-peer knowledge management and provides a concrete example of an application.
Contents

1 Introduction ............................ 1
   1.1 Technical contribution .................... 5
   1.2 Plan of the deliverable .................... 5

2 Modelling Secure Systems using an Agent-Oriented Approach and Security Patterns 6
   2.1 Security .................................. 6
   2.2 Secure Tropos ............................. 8
      2.2.1 The ”secure” concepts ................... 9
      2.2.2 Modelling Activities ..................... 10
      2.2.3 Integration to the current Tropos stages .......... 12
      2.2.4 An example ................................ 13
      2.2.5 Related Work ........................... 19
   2.3 Modelling Secure Systems using an Agent-Oriented Approach and Security Patterns ........ 22
      2.3.1 Agent Orientation and the Tropos Methodology .......... 22
      2.3.2 Using the Security Extensions of the Tropos Methodology .......... 24
      2.3.3 Security Patterns for Multiagent Systems ............ 27
      2.3.4 Applying the Security Patterns ............... 37
      2.3.5 Formalization of the Security Patterns .............. 39
      2.3.6 Related Work ........................... 43
   2.4 Security Patterns for Agent Systems ........... 47
      2.4.1 Selected Patterns of the Language ............. 49
   2.5 Towards the Development of Secure Information Systems: Security Reference Diagrams and Security Attack Scenarios .... 56
      2.5.1 Security Reference Diagram ............... 56
      2.5.2 Security Attack Scenarios ............... 61

3 Application of Security concepts .......... 68
   3.1 Using a Security Requirements Engineering Methodology in Practice: the compliance with the Italian Data Protection Legislation ........ 68
      3.1.1 The Italian Data Protection Legislation ............. 68
      3.1.2 University of Trento: Information System & Organization ...... 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.3</td>
<td>Security-Aware Tropos</td>
<td>71</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Modelling Actors</td>
<td>73</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Modelling Dependencies and Delegation</td>
<td>74</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Goal Refinement</td>
<td>75</td>
</tr>
<tr>
<td>3.1.7</td>
<td>Adequacy and Analysis of the Model</td>
<td>79</td>
</tr>
<tr>
<td>3.1.8</td>
<td>Related Case Studies and Conclusions</td>
<td>81</td>
</tr>
<tr>
<td>3.1.9</td>
<td>Comparing ISO-17799 and DPS 3.0</td>
<td>83</td>
</tr>
<tr>
<td>3.2</td>
<td>Using Tropos Methodology to Model an Integrated Health Assessment System</td>
<td>88</td>
</tr>
<tr>
<td>3.2.1</td>
<td>The single assessment process case</td>
<td>88</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Applying Tropos methodology</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>Multi-Agent Secure Interactions</td>
<td>102</td>
</tr>
<tr>
<td>4.1</td>
<td>Access control with Security Policies</td>
<td>102</td>
</tr>
<tr>
<td>4.2</td>
<td>A Running Example</td>
<td>102</td>
</tr>
<tr>
<td>4.3</td>
<td>Deduction vs Abduction</td>
<td>104</td>
</tr>
<tr>
<td>4.4</td>
<td>Stateful AC: Missing and Excessing Credentials</td>
<td>107</td>
</tr>
<tr>
<td>4.5</td>
<td>Life is complicated: Two-party Negotiation</td>
<td>109</td>
</tr>
<tr>
<td>4.6</td>
<td>Induction: Finding the rules</td>
<td>110</td>
</tr>
<tr>
<td>4.7</td>
<td>Implementation</td>
<td>111</td>
</tr>
<tr>
<td>4.8</td>
<td>The Challenges Ahead</td>
<td>112</td>
</tr>
<tr>
<td>5</td>
<td>Multi-Agent Databases</td>
<td>113</td>
</tr>
<tr>
<td>5.1</td>
<td>Towards Knowledge-Driven Peer-to-Peer computing</td>
<td>113</td>
</tr>
<tr>
<td>5.2</td>
<td>Towards Distributed Knowledge Management</td>
<td>115</td>
</tr>
<tr>
<td>5.3</td>
<td>Peer-to-Peer Knowledge Management - Prospects and an Application</td>
<td>116</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
<td>119</td>
</tr>
<tr>
<td>7</td>
<td>History of the Deliverable</td>
<td>121</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Agent Oriented Software Engineering (AOSE) introduces an alternative approach to an analysis and design of complex computerised systems [JW00, IGG99]. According to AOSE, a complex computerised system is viewed as a multiagent system in which autonomous software agents (subsystem) interact with each other in order to satisfy their design objectives. AOSE provides designers with more flexibility in their analysis and design. The actual design of the system takes place by specifying a multiagent system as a society, similar to a human society, consisting of entities that possess characteristics similar to humans such as mobility, and intelligence with the capability of communicating.

It has been argued, “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” [WJK00]. Thus, agent oriented software engineering have been developed to one of the most active research areas within the agent research community, and many methodologies have been developed. However, it is recognised amongst the agent research community [JW00, IGG99] the need of developing a complete methodology for analysing and designing multi-agent systems. The main role of such a methodology will be to help in all the phases of the development of a system, and more importantly, to help capture and model the unique characteristics that agent-oriented systems introduce such as flexibility, autonomous problem solving, and the rich interactions between the individual agents.

Security plays an important role in the development of multiagent systems and is considered as one of the main issues to be dealt for agent technology to be widely used outside the research community. As a result, research on security for Multiagent systems is an important area within the agent research community. However, the research has been mainly focused on the solution of individual security problems of the 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.

Only little work has been carried out to integrate security concerns into an agent-
oriented methodology. Developers of agent oriented software engineering methodologies
mainly neglect security. The common approach towards the inclusion of security within a
system is to identify security requirements after the definition of a system. This approach
has provoked the emergence of computer systems afflicted with security vulnerabilities
[Sta99a]. From the viewpoint of the traditional security paradigm, it should be possible to
eliminate such problems through better integration of security and systems engineering.
On the other side although it has been argued [DS00] that security concerns should inform
every stage of the development process, security is usually considered after the definition
of a multiagent system [MGM03d], leading to the development of systems afflicted with
security vulnerabilities [Sta99a].

A promising solution to this problem is to apply the pattern approach within the se-
curity domain. Patterns have proved their value for the development of software. One of
the main reasons that security is considered an afterthought is that software developers
do not know much about security and patterns could be an easy way to improve their un-
derstanding of security issues. A security pattern describes a particular recurring security
problem that arises in specific contexts and presents a well-proven generic scheme for its
solution. Moreover, the advantages of security patterns are that novices can rely on expert
knowledge and solve problems in a more structured way as, for instance, the dependencies
between problems can be identified and considered appropriately. Furthermore, security
experts can identify, name and discuss both problems and solutions more efficiently.

In this deliverable we introduce extensions to the Tropos methodology to accom-
modate security concerns during the software development stages [MGM03d, GM04,
MGM05, MWG05, MG04, BGM03, MGW03, MGM03b, MGM03f, MG06, BGMM04].
In particular, following the work in [WMG06], we propose the use of security patterns as
an extension to this methodology. Security patterns capture design experience and proven
solutions to security-related problems in such a way that can be applied by non-security
experts [FP01a]. Security patterns also prevent ad hoc solutions by helping apply proven
solutions in a systematic and structured way. In addition, they introduce abstraction layers
which help closing the gap between security experts and developers.

Computer systems constitute an inseparable part of our everyday life. Technologies
in computer systems advance rapidly and they are used in many different areas of human
society. Such an area is the Health Care sector. Health Care information systems are be-
coming more and more computerized. A huge amount of health related information needs
to be stored, analyzed and with the aid of computer systems this can be done faster and
more efficiently. One of the areas within the Health Care sector that can take advantage of
the computerizing of the health care systems, is the area related to the care of older peo-
ple. In a distributed health care setting different health care professionals, such as general
practitioners and nurses, must cooperate together in order to provide older persons with
appropriate care and must also work closely with social care professionals, such as social
workers, because health and social care needs overlap amongst older people. Computer-
izing this process will help to automate some of the tasks of the health and social care professionals and thus leave the professionals with more time for the care of the older people.

We present an electronic Single Assessment Process (eSAP), an electronic system to deliver an integrated assessment of health and social care needs of older people. From a variety of different analysis and design methodologies for agent-based system, we have identified the Tropos methodology (for more detail see [ea06]) to proceed with the analysis and design of the eSAP system.

Again the last years have seen a major interest in the development of requirements engineering (RE) methodologies which are able to capture security requirements. This has been marked by some workshops (SREIS, SAPS, REHAS, et al.) and many papers and books [AE04, LYM03, GMMZ04b, SO00b, vBDJ03, Jür04]. We present a major case study of the application of the Secure Tropos requirements engineering modelling and formal analysis methodology [GMMZ04b, GMMZ04a] for the compliance to the Italian legislation on Privacy and Data Protection by the University of Trento.

From another point of view controlling access to services is a key aspect of networking and the last few years have seen the domination of policy-based access control. Indeed, the paradigm is broader than simple access control, and one may speak of policy-based network self-management (e.g. [SL99] or the IEEE Policy Workshop series). The intuition is that actions of nodes “controlling” the communication are automatically derived from policies. Nodes look at events and requests presented to them, evaluates the rules of their policies and derive actions [SL99, Smi03]. Policies can be “simple” iptables rules for Linux firewalls (see http://www.netfilter.org/) or complex logical policies expressed in languages such as Ponder [DDLS01].

Autonomic Communication adds new challenges: a truly autonomic network is born when nodes are no longer within the boundary of a single enterprise which could deploy its policies on them and guarantee interoperation. Nodes are partners that offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross-enterprise scenario poses novel security challenges with aspects of both trust management systems and workflow security.

From trust management systems [Wee01, EFL99, LGF03] it takes the credential-based view: access to services is offered by autonomic nodes on their own and the decision to grant or deny access must rely on attribute credentials sent by the client. In contrast with these systems, we have a continuous process and assignment of permissions to credentials must look beyond the single access decision.

From workflow access control [ACM01, BFA99, GHS95, KPF01] we borrow all classical problems such as dynamic assignment of roles to users, separation of duties, and assignment of permissions to users according the least privilege principles. In contrast with such schemes, we can no longer assume that the enterprise will assign tasks and
roles to users (its employees) in such a way that makes the overall flow possible w.r.t. its security constraints.

Following the work in [KM04a, KM05a, KM05b, KM04c, KM04b, KM04d, KM03c, KM03b, KM03a] we sketch the reasoning framework for access control for autonomic communication based on interaction for supplying missing credentials or for revoking “wrong” credentials. We identify the reasoning services deduction vs abduction, and induction, and sketch the solution for stateful access control and mutual negotiation. A running example makes discussion concrete.

Peer-to-Peer (P2P) computing has received significant attention from the side of research labs and academia, largely due to the popularity of commercialized P2P file sharing applications such as Napster, Morpheus and KaZaa. In the P2P model, peers exchange data and/or services in a completely decentralized distributed manner. Peers are autonomous, and are free to choose what other peers to interact with. In this point-to-point interaction, peers possess equal functional capabilities.

On the other hand, Knowledge Management (KM) is increasingly viewed as a core capacity in order to compete in the modern social and economic environment [TL00, P.90, IH95]. Researchers and practitioners agree that those intellectual assets [T.A02] that are embedded in working practices, social relationships, and technological artefacts constitute the only source of value that can sustain long term differentiation, quality of services, innovation, and adaptability. Nonetheless, even due to a debatable success of current KM implementations, it is still unclear how such matter should be managed in highly complex, distributed, and heterogeneous settings.

In the last couple of years, P2P and KM have followed different but converging paths. In fact, P2P technologies have left their initial "computational", "anarchoyd", and spontaneous fashion to embrace more service level domains and business settings. On the other hand, KM is questioning its centralized assumption based on the implicit belief that knowledge is managed successfully when it can be standardized and controlled. In this sense, it seems that while P2P is looking for value added domains to better exploit its technological potential, KM is looking for a technological paradigm more able to fit an emerging distributed organization of knowledge.

Finally, this deliverable gives a short overview of the P2P model and applications, and explains the increasing requirement for P2P applications to address knowledge-driven domains. After a short introduction to the KM field, and description of the advantages of distributed KM solutions, we discuss what synergies can benefit integrated P2P KM solutions, and present a concrete example of a P2P KM application.
1.1 Technical contribution

In this deliverable we introduce extensions to the Tropos methodology to enable it to model security concerns throughout the whole development process. A description of the new concepts is given along with an explanation of how these concepts are integrated to the current stages of Tropos. Specifically, as an extension to the existing methodology we propose the use of security patterns. These patterns capture proven solutions to common security issues, and support the systematic and structured mapping of these constraints to an architectural model of the system, in particular for non-security specialists. We present a pattern language for secure agent systems that, currently, consists of four patterns.

We discuss case studies to illustrate the features and the stages of the Tropos methodology. In particular, the electronic Single Assessment Process (eSAP), an electronic system to deliver the integrated health assessment of health and social care needs of older people is used as a case study. Also, we present a comprehensive case study of the application of the Secure Tropos RE methodology for the compliance to the Italian legislation on Privacy and Data Protection by the University of Trento, leading to the definition and analysis of a ISO-17799-like security management scheme.

We propose a logical framework and a Web-Service based implementation for reasoning about access control for Autonomic Communication. Our model is based on interaction and exchange of requests for supplying or declining missing credentials. We identify the formal reasoning services that characterise the problem and sketch their implementation.

Finally, we discuss the state of the art and trends in both the P2P and KM fields, as well as what possible synergies can benefit integrated P2P KM solutions, and present an implemented P2P KM system.

1.2 Plan of the deliverable

The rest of the deliverable is organized as follows. Chapter 2 concentrates on modelling secure systems by using agent-oriented approach and security patterns. Chapter 3 provides two case studies on using a security requirements engineering methodology in practice. Chapter 4 is devoted to a logical framework for reasoning about access control for autonomic communication. Finally, Chapter 5 discusses the issues in peer-to-peer knowledge management as well as a concrete example of a P2PKM application.
2.1 Security

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.

According to the Computer Crime and Security Survey, contacted by the Computer Security Institute (CSI) during the first three quarters of 2002, about ninety (90) percent of respondents, mainly large US corporations and US government agencies, detected
computer security breaches and eighty (80) 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\(^1\) while during 1995, 171 vulnerabilities were reported, this number increased to 3,222 during the first three quarters of 2002. In addition, the last 10 years the number of incidents reported has increased from 773 (in 1992) to 73,359 (the first three quarters of 2002).

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. Thus, currently the definition of security requirements is usually considered after the design of the 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.

The same is true for Multiagent systems. A main reason for this situation is that developers of agent-oriented methodologies have mainly neglected security. Although, many agent oriented methodologies have been developed during the last few years [IGG99, CKM01a], very little evidence have been reported of methodologies that they adequately integrate security and systems engineering within their development process. Agent Oriented software engineering considers security as a non-functional requirement. However, differently than other non-functional requirements, such as performance and reliability, the definition of security is usually considered after the design of the system.

We believe that security concerns should be considered during the whole development process of a Multiagent system and it should be defined together with the requirements specification. Taking security requirements into account together with the functional requirements of a Multiagent system throughout the development stages helps to limit cases of conflict between security and system requirements, 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).

The integration of security concerns within the context of a multiagent system will require for the subsystems (agents) of the system to consider the security requirements when specifying their goals and interactions therefore causing the propagation of security requirements to the rest of the subsystems.

\(^1\)http://www.cert.org/
2.2 Secure Tropos

Tropos [CKM01a] is a software development methodology, for building agent-oriented software systems, that uses concepts such as actors, goals, soft goals, tasks, resources (see figure 2.1 for graphical representation) and intentional dependencies (see figure 2.2 for graphical representation) throughout all the phases of the software development [PBG+01b]. A key feature of Tropos is that it pays great deal of attention to the early requirements analysis that precedes the specification of the perspective requirements, emphasizing the need to understand the how and why the intended system would meet the organisational goals.

![Figure 2.1: Graphical Representation of Tropos concepts.](image)

Tropos supports four development stages, namely early and late requirements, architectural design, and detailed design. Early and late requirements analysis represents the initial phases in the Tropos methodology and the final goal is to provide a set of functional and non-functional requirements for the system-to-be. Both phases, early and late, share the same conceptual and methodological approach. This means, that most of the techniques used during the early requirements analysis are used for the late as well. The main difference is that during the early requirements analysis, the developer models the main stakeholders of the system and their dependencies, while in the late requirements analysis the developer models the system itself by introducing it as another actor and model its dependencies with the other actors of the organisation. The architectural design stage defines the system’s global architecture in terms of actors interconnected through data and control flows (represented as dependencies). In addition, during this stage the actors of the system are mapped into a set of software agents, each characterized by its specific capabilities. During the detailed design stage, the developer specifies, in detail, the agents’ goals, beliefs, and capabilities as well as the communication between the agents. For this reason, Tropos employs a set of AUML diagrams [BMO01a].

Tropos was not conceived with security in mind and as a result it fails to adequately capture security requirements [MGMP02b]. The process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc, and in addition, the concept of soft goal that Tropos uses to capture security requirements fails to adequately capture some constraints that security requirements often represent.
Thus, we have extended the Tropos methodology to enable developers to adequately capture security requirements. The next section describes our extensions and how they have been integrated within the Tropos methodology process.

2.2.1 The “secure” concepts

Extra concepts were introduced to the methodology to enable it to model security requirements during the software development process. These are:

**Security Diagram** [Mou02], that represents the connection between security features, threats, protection objectives, and security mechanisms that help towards the satisfaction of the objectives. Security features [Mou02] represent security related features that the system-to-be must have. Protection objectives [Mou02] represent a set of principles that contribute towards the achievement of the security features. Threats [Mou02] on the other hand represent circumstances that have the potential to cause loss or problems that can put in danger the security features of the system, while security mechanisms [Mou02] identify possible protection mechanisms of achieving protection objectives.

**Security Constraint** [MGMP02b], which represents constraints that are related to the security of the system. Since, constraints can influence the security of the system either positively or negatively, we further define positive and negative security constraints respectively. An example of a positive security constraint could be Allow Access Only to Personal Information, while a negative security constraint could be Send Information Plain Text (not encrypted).

**Secure Entities** [MGMP02b], which represent any secure goals/tasks/resources of the system. Secure goals are introduced to satisfy possible security constraints that exist in the system, while security tasks represent ways of achieving the introduced security goals. A resource that is related to a secure entity or a security constraint is considered a secure resource.

**Secure Dependencies** [MGMP02b], represent that a dependency between two actors involves the introduction of a security constraint that must be satisfied either by the depender, the dependee or both for the dependency to be valid. Secure dependencies are categorized into depender secure dependency, in which the depender introduces security constraints for the dependency and the dependee must satisfy the security constraints for the dependency to be valid, dependee Secure Dependency, in which the dependee introduces security constraints and the depender must satisfy them, and double Secure Dependency, in which both the depender and the dependee introduce security constraints for the dependency that both must satisfy for the depen-
dency to be valid. A graphical representation of the different types of dependencies is illustrated in figure 2.2.

Secure Capabilities, which represent capabilities that the actors (agents) of the system must have in order to help towards the satisfaction of the security requirements of the system.

2.2.2 Modelling Activities

Many different modeling activities contribute to the capture of security requirements and their integration within the Tropos development process. These are:

Security diagram modelling, which involves the modelling of security needs of the system-to-be, problems related to the security of the system (such as possible threats and vulnerabilities) and possible solutions to the security problems (these solutions
can usually be identified in terms of a security policy that the organisation might have).

We have decided to use Tropos concepts to capture the concepts of the security diagram. This helps to better integrate the security diagram within the Tropos methodology and make its concepts easily understandable to developers familiar with Tropos. Thus, we are using the concept of a soft goal to capture security features. Soft goals in Tropos are used to model quality attributes for which there are no clear criteria for satisfaction. In the same sense, security features are not subject to any clear criteria for satisfaction. Protection objectives are represented using the concept of a goal, because as goals define desired states of the world, protection objectives define desired security states that the system must have. To represent security mechanisms we use the concept of a task, since as a task represents a way of doing something (usually a goal), a security mechanism represents a way of achieving a security objective. The following figure shows how the above-mentioned concepts can be graphically represented.

![Security diagram concepts](image)

**Figure 2.3: Security diagram concepts.**

**Security constraint modelling** involves the modeling of the security constraints imposed to the actors and the system and it allows the designer to perform an analysis by introducing relationships between the constraints or a constraint and its context [Mou02]. Security constraints are imposed by the stakeholders (during the early requirements stage) and by the security diagram (during the late requirements stage) and are guaranteed by assigning capabilities (secure capabilities) to the components of the system (i.e. the actors or the agents of it). Stakeholders can impose positive and negative security constraints, while the constraints imposed by the security diagram are only positive security constraints. By imposing security constraints to different parts of the system, we are able to identify possible conflicts between security and other (functional and non functional) requirements of the system, identify (stakeholder) constraints that can put in danger the security of the system, and propose possible ways towards a design that will integrate security and systems engineering leading to the development of a more secure system. A security constraint is represented graphically as shown in figure 2.4.

**Secure entities modelling**, which is considered as complementary to the security constraints modeling. The analysis of the secure entities follows the same reasoning techniques identified by Tropos for the goal and task analysis [BPG+02]. Secure
Entities are represented by introducing an S within brackets (S) before the text description as shown in figure 2.4.

![Security Constraint](image1)

- **Secure goal**
- **Secure resource**
- **Secure task**

Figure 2.4: Security constraint and secure entities graphical representations.

**Secure capability modelling**, involves the identification of the capabilities of the subsystems (actors) to guarantee the satisfaction of the security constraints. In order for the system to be able to satisfy its goals and the security constraints, the system’s agents has to be provided with capabilities. Secure Capabilities modelling takes place alongside 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. Later, during the detailed design, these capabilities are further specified (in terms of plans, etc) and they are coded during the implementation stage.

### 2.2.3 Integration to the current Tropos stages

A difficult but necessary task when extending a methodology is to successfully integrate the extensions to the current stages of the methodology. In our case, we have integrated the extensions as follows:

**Early requirements stage**: During the early requirements analysis stage the Security Diagram (SD) is constructed and security constraints are imposed to the stakeholders of the system (by other stakeholders). However, the imposed security constraints are expressed in high-level statements, so they are furthered analysed [Mou02] and security entities are introduced to satisfy them.

**Late requirements stage**: During the late requirements stage security constraints are imposed to the system-to-be (by the security diagram). These constraints are further analysed according to the constraint analysis processes [Mou02].

**Architectural design stage**: During the architectural design we identify the security constraints and secure entities that the new actors introduce and also during the actor decomposition we identify security sub-constraints and sub-entities. In addition secure capabilities are identified and assigned to each agent of the system.

**Detailed design stage**: We specify the agent capabilities and interactions taking into account the security aspects as well. In doing so we are using AUML [BMO01a]
notation in which we introduce the tag of security rules. This is similar to the business rules that UML has for defining constraints on the diagrams.

2.2.4 An example

In this section, we go through the development stages using a case study. This case study is part of a real-life system, called electronic Single Assessment Process (eSAP), under development at the University of Sheffield [MiPM02]. The electronic Single Assessment Process (eSAP) system is an agent-based health and social care system for the effective care of older people. To make this example simpler and more understandable, we consider a substantial part of the eSAP system.

Early Requirements

Early Requirements stage is concerned with the understanding of a problem by studying an existing organisational setting. The output of this phase is an organisational model, which includes relevant actors and their respective dependencies. In our example, we consider the following actors for the eSAP system:

- Professional: The health and/or social care professional
- Older Person: The Older Person (patient) that wishes to receive appropriate health and social care
- DoH: The English Department of Health
- R&D Agency: A Research and Development Agency interested in obtaining medical information
- Benefits Agency: An agency that helps the older person financially

The first step in the early requirements analysis is the construction of the security diagram. The main security features of the security diagram for the eSAP system are privacy, integrity and availability. However, for our example we consider only two desired security feature, namely privacy and availability. A part of the security diagram, taking into account privacy and availability is shown in figure 2.5.

The next step involves the modelling of goals, dependencies and security constraints between the stakeholders (actors). For this purpose we are employing actors’ diagram. In such a diagram each node represents an actor, and the links between the different actors indicate that one depends on the other to accomplish some goals. In addition, the imposed security constraints (by other stakeholders) indicate that the actors must satisfy them for the dependencies to be valid. For example, the Older Person depends on the Benefits Agency to Receive Financial Support. However, the Older Person worries about
the privacy of their finances so they impose a constraint to the Benefits Agency actor, to keep their financial information private. The Professional depends on the Older Person to Obtain (Older Person) OP Information, however one of the most important and delicate matters for the older person (as with any patient) is the privacy of their personal medical information, and the sharing of it. Thus, most of the times, the Professional is imposed a constraint to share this information if and only if consent is achieved. 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. However, the Professional is imposed a constraint (by the Department of Health) to Keep Patient Anonymity. Figure 2.6 illustrates part of the actor diagram of the eSAP system taking into consideration the above-mentioned constraints that are imposed to the stakeholders of the system.

When the stakeholders, their goals, the dependencies between them, and the security constraints have been identified, the next step of this phase is to analyse in more depth each actor’s goals and the security constraints imposed to them. In addition, secure entities are introduced to help towards the satisfaction of the imposed security constraints.

The analysis of the security constraints starts by identifying which goals of the actor they restrict. The assignment of a security constraint to a goal is indicated using a constraint link (a link that has the "restricts" tag). For example, the Professional actor (figure 2.7) has been imposed two security constraints (Share Info Only If Consent Achieved and Keep Patient Anonymity). During the means-end analysis of the Professional actor
we have identified the Share Medical Info goal. However, this goal is restricted by the Share Info Only If Consent Achieved constraint imposed to the Professional by the Older Person. For the Professional to satisfy the constraint, a secure goal is introduced Obtain Older Person Consent. However this goal can be achieved with many different ways, for example a Professional can obtain the consent personally or can ask a nurse to obtain the consent on their behalf. Thus a sub-constraint is introduced, Only Obtain Consent Personally. This sub constraint introduces another secure goal Personally Obtain Consent. This goal is divided into two sub-tasks Obtain Consent by Mail or Obtain Consent by Phone. The Professional has also a goal to Provide Medical Information for Research. However, the constraint Keep Patient Anonymity has been imposed to the Professional, which restricts the Provide Medical Information for Research goal. As a result of this constraint a secure goal is introduced to the Professional, Provide Only anonymous Info.

Late Requirements

When all the actors have been analysed, the next phase involves the analysis of the system-to-be. During the late requirements stage, the system-to-be is analysed within its operation
environment, along with relevant functions, security concerns and qualities. The system is presented as one or more actors, who have a number of dependencies with the other actors of the organisation. These dependencies define all the functional and non-functional requirements of the system. However, in this example, we are focusing on the security modeling. From the security point of view, security constraints are imposed to the system-to-be (by taking into account the security diagram). These constraints are further analysed according to the constraint analysis processes [Mou02].

The main aim of the eSAP system (figure 2.8) is to Automate Care in order to help professionals provide faster and more efficient care, and allow on the other hand older people get more involved in their care. Taking into consideration the security diagram there are two main constraints imposed (by the desired security features of the system—privacy and availability) to the eSAP’s main goal - Keep Data Private and Keep Data Available. For the eSAP to satisfy these constraints two secure goals have been identified. Ensure Data Privacy and Ensure Data Availability.

This example focuses only on the Keep Data Private constraint. This constraint can be further analysed to sub-constraints Allow Only Encrypted Transfer of Data, Allow Only Authorised Access, and Allow Access Only to Personal Care Plan. Taking into account the security diagram, secure goals are introduced to help towards the satisfaction of the imposed security constraints. Thus the secure goals Use Cryptography, Check Authorisation, Check Access Control, and Check Information Flow are introduced. In addition, some of the secure goals are further analysed in terms of secure tasks.
Thus, the Use Cryptography goal is divided to two secure tasks Encrypt Data and Decrypt Data. Although these tasks could be furthered decomposed by indicating for example the type of the encryption algorithm this is not the case in this stage, since the type of the encryption algorithm depends on the implementation of the system and it will restrict the designers of the system in a particular implementation style. The Check Authorisation is decomposed into four secure tasks, Check Password, Check Digital Signatures, Check Biometrics and Call Back. However, it is indicated in the diagram that the last two tasks contribute negatively towards the mobility of the system, and this is one factor that the developers must take into consideration in the implementation of the system.

**Architectural Design**

The architectural design involves the addition of new actors, in which new actors are added to make the system interact with the external actors; actor decomposition, in which each actor is described in detail with respect to their goals and tasks; capabilities identification, in which capabilities needed by the actors to fulfill their goals are identified; and agent assignment, in which a set of agent types is defined and each agent is assigned one or more capabilities. From the security point of view, we identify the security constraints and secure entities that the new actors introduce and also during the actor decomposition we identify security sub-constraints and sub-entities. In addition secure capabilities are identified and assigned to each agent of the system.

As it was derived from the late requirements stage, one of the systems secure goals is
to "Ensure Data Privacy". To achieve this goal the eSAP depends on the Privacy Manager (Figure 2.9).

The Privacy Manager has four main secure goals, as derived from the analysis, (sub-goals to the "Ensure Data Privacy" goal) "check authorisation", "check access control", "check information flow" and "use cryptography". For achieving these goals the Privacy Manager depends on the "Authorisation Manager", "Access Control Manager", "Information Flow Manager" and "Cryptography Manager" respectively. Although the other actors of the system can be further decomposed, due to lack of space, this example focuses only in the privacy concerns of the system. For each new actor introduced in the system, an extended diagram is required to capture the dependencies of the new actor with the already existing actors of the system. Figure 2.10 shows a part (focused on the privacy) of the extended diagram for the plan "Access Care Plan Info" of the Professional. The Care Plan Manager is responsible for providing access at the Professional to "Care Plan Info". It depends on the Authorisation Manager to deal with authorisation procedures, on the Access Control Manager and the Information Flow Manager to perform access control checks and information flow checks respectively, and on the Cryptography Manager for encrypting and decrypting information.

The next step in the architectural design is to identify (secure) capabilities for each actor. Taking into consideration the extended actor diagram (figure 2.10), each dependency relationship can give place to one or more capabilities triggered by external events. The
actors along with their capabilities with respect to the extended diagram of figure 2.10 are shown in Table 2.1. When the actors along with their capabilities have been identified the next step is the agents’ assignment. A set of agent types are defined and each one of them is assigned one or more different capabilities (Table 2.2) with respect to the capabilities identified in the previous step (Table 2.1).

**Detailed Design**

From the security point of view, during the detailed design the developers specify the agent capabilities and interactions taking into account the security aspects derived from the previous steps of the analysis. In doing so AUML notation is employed. The only difference is the introduction of security rules. These are similar to the business rules that UML has for defining constraints on the diagrams.

![Extended Diagram wrt "Access Care Plan Info" task.](image)

**2.2.5 Related Work**

As stated in the introduction, very little work has taken place in considering security requirements as an integral part of the whole software development process. None of the existing agent oriented methodologies, to our knowledge, have been demonstrated enough evidence to support claims of adequately integrate security modeling during the whole software development stages. Only recently, some initial steps have been taken towards
<table>
<thead>
<tr>
<th>Actors</th>
<th>Capability</th>
<th>Cap .ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professional</td>
<td>Provide Care Plan Info Request</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Provide Authorisation Details</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Obtain Care Plan Info</td>
<td>3</td>
</tr>
<tr>
<td>Care Plan Manager</td>
<td>Obtain Care Plan Info Request</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Provide Care Plan Info</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Request Encryption of Data</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Obtain Encrypted Data</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Request Decryption of Data</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Obtain Plain Data</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Obtain Authorisation Clearance</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Obtain Access Control Clearance</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Obtain Information Flow Clearance</td>
<td>12</td>
</tr>
<tr>
<td>Cryptography Manager</td>
<td>Encrypt Data</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Decrypt Data</td>
<td>14</td>
</tr>
<tr>
<td>Information Flow Manager</td>
<td>Provide Information Flow Clearance</td>
<td>15</td>
</tr>
<tr>
<td>Access Control Manager</td>
<td>Provide Access Control Clearance</td>
<td>16</td>
</tr>
<tr>
<td>Authorisation Manager</td>
<td>Obtain Authorisation Details</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Provide Authorisation Clearance</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.1: Actors and their Capabilities
this direction. Eric Yu has initiated work [LYM02b] that provides ways of modeling and reasoning about non-functional requirements (with emphasis on security). Yu is 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.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professional</td>
<td>1,2,3</td>
</tr>
<tr>
<td>Care Plan Agent</td>
<td>4,5,6,7,8,9,10,11,12</td>
</tr>
<tr>
<td>Privacy Agent</td>
<td>13,14,15,16,17,18</td>
</tr>
</tbody>
</table>

Table 2.2: Agents and their Capabilities

Lodderstedt et al present a modeling language, based on UML, called SecureUML [LBD02]. Their approach is focused on modeling access control policies and how these (policies) can be integrated into a model-driven software development process. Differently than these two approaches that are focused in particular stages of the development (Yu’s effort is focused only in the requirements area while Lodderstedt’s work is focused in the design stage) our approach covers the whole development process. It is important to consider security using the same concepts and notations during the whole development process.

In addition, Huget [Hug02] proposes a new 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 such it has not been extensively presented on literature. From our point of view, the methodology tackles security quite superficial and as the developer states “particularly, security has to be intertwined more deeply within models” [Hug02]. Thus, more evidence will be required to satisfy the claim of the developer that the methodology tackles security.
2.3 Modelling Secure Systems using an Agent-Oriented Approach and Security Patterns

2.3.1 Agent Orientation and the Tropos Methodology

Agent Oriented Software Engineering (AOSE) is emerging as a powerful, new paradigm for the development of information systems. Its major modeling construct, agents, demonstrate properties such as autonomy, intentionality, sociality, identity and boundaries, strategic reflectivity, and rational self-interest\cite{Yu02}. As a result, agent orientation provides a higher level of abstraction than previous software development paradigms, such as object orientation. However, an important point of agent-oriented software engineering is that using it for the analysis and design of a system does not necessarily impose the use of agents as an implementation choice.

Tropos is a development methodology tailored to the description of the organisational environment of a system and the system itself\cite{BGG04}. Tropos emphasizes the early requirements analysis that precedes requirements specification, addressing the need to understand how and why the intended system would meet the organisational goals. This allows for a more refined analysis of 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.

Tropos adopts the $i^*$ modelling framework, which uses the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations of actors (dependees) to other actors (dependers)\cite{Yu95}. Actors have strategic goals and intentions within the system or the organisation and represent (social) agents (organisational, human or software), roles or positions (a set of roles). A goal represents the strategic interests of an actor. We differentiate between hard goals (or simply goals) and soft goals. Soft goals represent non-functional requirements and have no clear definition or criteria for deciding on whether they are satisfied or not. An example of a soft goal is “the system should be scalable”. A task represents a way of doing something. Thus, for example a task can be executed in order to satisfy a goal. A resource represents a physical or an informational entity, while a social dependency between two actors indicates that one actor depends on another to accomplish a goal, execute a task, or deliver a resource. Figure 2.11 shows the notation for these concepts.

Although Tropos was not conceived with security in mind, a set of security concepts, such as security constraint, secure entities and secure dependencies have been proposed by Mouratidis that allow Tropos to model security aspects throughout the full development process\cite{MGMP02a, MGM03a}. A security constraint is defined as a constraint that is related to the security of the system, while secure entities represent any secure goals/tasks/resources of the system. Secure goals are introduced to the system to help in the achievement of a security constraint. A secure goal does not define specifically how the security constraint can be achieved, since (as in the definition of a goal) alternatives
Figure 2.11: Notation used by the Tropos modelling framework. It uses the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations between actors.

...can be considered. However, this can be modeled as a secure task, which secure task represents a particular way for satisfying a secure goal.

A secure dependency introduces security constraints, proposed either by the depender or the dependee in order to successfully satisfy the dependency. Both the depender and the dependee must agree on these constraints for the dependency to be valid. The depender expects the dependee to satisfy the security constraints, while the dependee will make an effort to deliver the dependum by satisfying the constraints. The security concepts added to Tropos are shown in Figure 2.12.

Figure 2.12: Extensions to the Tropos notation for modelling security concepts.

Tropos covers five main software development phases:[BGG⁺04a]

- early requirements analysis, concerned with the understanding of a problem by studying an existing organisational setting;
- late requirements analysis, where the system is described in its operating environ-
ment, along with relevant functions and security requirements;

- architectural design, where the global system architecture is defined in terms of subsystems, interconnected through data and control flows;

- detailed design, where each architectural component is defined in terms of inputs, outputs, control, and security aspects; and

- implementation, during which the system components are implemented according to the previous phases (not necessarily agent-based).

### 2.3.2 Using the Security Extensions of the Tropos Methodology

To demonstrate how the above security concepts and procedures can be used in the development of secure information systems, we consider the electronic Single Assessment Process (eSAP) system, an integrated health and social care information system for the effective care of older people[MPM02a]. Security is an important concern for eSAP, since security breaches of such a system might result in personal and health information to be revealed, which could lead to serious consequences.
Early Requirements Analysis

During the early requirements analysis phase, the goals, dependencies and the security constraints between the stakeholders (actors) are modeled with the aid of an actor diagram[1]. Such a diagram involves actors and the dependencies between the actors. Some of the actors involved in the eSAP system along with their dependencies and security constraints are shown in Figure 2.13.

However, an actor diagram does not provide an analysis of the individual actor’s goals and the security constraints imposed on them. This information can be modelled with the aid of rationale diagrams[1]. In a rationale diagram, an actor’s goals and security constraints are explicitly analysed. A dashed circle encloses the actor. Figure 2.14 illustrates the rationale diagram of the Older Person actor.

Late Requirements Analysis

Once the actors have been analysed and their goals, dependencies and security constraints identified, we can proceed to the late requirements analysis phase. In this phase, the functional, security, and other non-functional requirements for the system to be are elaborated. The system to be is introduced as one or multiple actors that have a number of dependen-
cies with the other actors of the organization (as defined during the early requirements phase). For example, the eSAP system contributes to the goals of the stakeholders as shown in Figure 2.15.

The introduced system (the eSAP system in our case study) is further analysed using the same concepts used for the analysis of the other actors. Figure 6 shows a rationale diagram of the eSAP system. To satisfy the security objectives of the system, different security constraints are imposed. In our example, the security constraints have been derived from the security policy for medical information systems identified by Anderson [And01a]. Then, the security constraints are further analysed according to security constraint analysis [MGMP02a, MGM03a].

For instance, by analysing the Keep System Data Private security constraint in Figure 2.16, we derive that tasks such as Check Access Control, Check Authentication, and Check Information Flow must be achieved in order to fulfill this security constraint. Each of those tasks can be achieved by considering different alternatives. For example, Check Authentication can be achieved in three different ways: Check Password, Check Digital Signature, or Check Biometrics.
2.3.3 Security Patterns for Multiagent Systems

Using the security-oriented extension of the Tropos methodology we have identified the security requirements of the system during early and late requirements analysis. The next phase of the methodology is the architectural design phase. During this phase, the requirements are transformed into a design. However for a developer without knowledge of security this could be a very difficult task, possibly resulting in the development of a non-secure system. For this reason we introduce security patterns. Patterns capture existing proven experience in software development and help to promote best design practices[Cop96].

Patterns are often organized in the form of pattern languages. 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 realized. 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[BJ94].

A pattern language for the development of secure agent-based systems should employ agent-oriented concepts, such as intentionality, autonomy, sociality and identity. The structure of a pattern should be described not only in terms of collaborations and the message exchange between the agents, but also in terms of their social dependencies and intentional attributes, such as goals and tasks. This allows for a complete understanding of the pattern’s social and intentional dimensions.

Our pattern language has four patterns: Agency Guard, Agent Authenticator, Sandbox, and Access Controller. For each pattern we provide its name and context, the problem to which it is applicable, the solution to this problem, the social dependencies, and the consequences of applying the pattern. Figure 2.17 provides a roadmap of the pattern language that indicates how the patterns relate to one another. The arrows in the roadmap show dependencies between patterns, and point from one pattern to the patterns that developers may want to consult once this pattern has been applied. The roadmap thus suggests to begin the architectural design with the Agency Guard pattern. The annotations on the arrows summarize the rationale for selecting a pattern in the context of another pattern.

We use the Alexandrian format for organizing each pattern[AIS77]. The sections of a patterns are context, problem and forces, solution, and rationale, each section set off from the next by a set of stars. Brief descriptions of the problem and solution are put in boldface, followed by more detailed discussions. The rationale section is organized into benefits, liabilities, and related patterns.

Agency Guard

... a number of agencies exist in a network. Agents from different agencies must communicate with each other, or exchange information. This involves the movement of some
Figure 2.16: Rationale diagram for the eSAP system.
agents from one agency to another, or requests from agents belonging to one agency for resources belonging to another agency.

***

A malicious agent that gains unauthorized access to the agency can disclose, alter or generally destroy data residing in the agency.

Many malicious agents will try to gain access to agencies that they are not allowed to access. Depending on the level of access the malicious agent gains, it might be able to completely shut down the agency, or exhaust the agency’s computational resources, and thus deny services to authorised agents. The problem becomes the more severe the more backdoors there are to an agency, enabling potential 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 rather access should be granted based on the security policy of the agency.

Therefore:

**Ensure that there is only a single point of access to the agency.**

When a Requester Agent wishes to access resources of an Agency or move to this agency, its requests must be forwarded through an Agency Guard that is responsible for granting or denying access requests according to the security policy of the agency. The Agency Guard is the only point of access to the Agency, and cannot be bypassed, meaning all access requests must go through it. In traditional terms, the concept of an Agency Guard is referred to as a monitor[Amo94a].

The structure of the pattern in terms of the actors involved and their social dependencies is shown in Figure 2.18. The remaining dependencies are as follows. The Agency

---

Figure 2.17: Roadmap of the pattern language for secure agent systems
Figure 2.18: Structure of the *Agency Guard* pattern.

depends on the *Agency Guard* to grant or deny access to the *Agency*. The *Agency Guard* will grant or deny access according to the security policy of the *Agency*, and depends on the *Agency* to obtain this security policy.

***

**Benefits:**

- Only the *Agency Guard* needs to be aware of the security policy, and it is the only entity that must be notified if the security policy changes.

- Being the single point of access, only the *Agency Guard* must be tested for correct enforcement of the agency’s security policy.

**Liabilities:**

- A single point of access to the agency can degrade the performance of the agency (that is, the response time for handling access requests).

- The *Agency Guard* is a single point of failure. If it fails, the security of the agency as a whole is at risk.

**Related patterns:**

- *Agent Authenticator* – ensures the identity of the *Requester Agent*.  

30
Agent Authenticator

... you are using Agency Guard to protect access to an agency or its resources. To be allowed access, agents must be authenticated, that is, they must provide information about the identity of their owners.

***

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 under attack will no longer trust the agent impersonated by the malicious agent.

Therefore:

Authenticate agents as they enter the agency.

Requester Agents have to be authenticated by the Agency. By authenticating the agent, the Agency Guard makes sure it comes from an owner that is trusted by the Agency. Each Requester Agent’s owner and each Agency have a public/private key pair. The Agent Authenticator can authenticate the Requester Agent in two ways: the agent can be digitally signed with the owner’s public key, or with the private key of the Agency in which the agent resides. In order for the second approach to work, mutual trust must be established between the sending and receiving agencies (each Agency can be set up so it has a list of trusted agencies). If the Agent Authenticator does not trust the Agency from which the agent originates, it can reject the agent, or accept it with minimal privileges, and be executed in a Sandbox.

The structure of the pattern is shown in Figure 2.19. The Agency Guard depends on the Agent Authenticator to authenticate the agent, and, in turn, the Agent Authenticator has to receive the request from the Agency Guard. The Agent Authenticator has to send the notification to the Agency Guard once the agent is authenticated.

***

Benefits:

- Since authentication concerns are dealt with in a single location, it is not necessary to provide each agent with its own authentication mechanism.

- The use of an Agent Authenticator ensures that Requester Agents are authenticated, before they can request a resource from the agency.
Figure 2.19: Structure of the Agent Authenticator pattern.

- When implementing the system, only the Agent Authenticator must be checked for correct enforcement of the agency’s security policies.

Liabilities:

- The Agent Authenticator is a single point of failure. If it fails, the security of the agency as a whole is at risk.

Related patterns:

- Sandbox – allows running an agent that could not be authenticated with minimal privileges.
- Access Controller – restricts access to the agency’s resources.

Sandbox

... you are using Agent Authenticator to ensure the requester agent’s identity, but the requester agent cannot be properly authenticated. This can be the case either when the
agent could not be authenticated, or if it has been authenticated by an agency that the
receiving agency does not trust.

***

**An agency is most likely exposed to a large number of malicious agents that will try
to gain unauthorized access to it.**

Although the agency will try to prevent access to those agents, it is possible that some
of them might be able to gain access to the agency’s resources. Thus, it is necessary for
the agency to operate in a manner that will minimize the damage which can be caused
by unauthorized agents gaining 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 unauthorized agents without
putting its security at risk.

Therefore:

**Execute the agent in an isolated environment that has full control over the agent’s
ingoing and outgoing messages.**

Implementing this principle prevents any malicious code from doing something it is not
authorised to do. The code is allowed to destroy anything within a restricted environment
(a sandbox), but it cannot touch anything outside. The concept is similar to the Java se-
curity model, and the chroot environment in UNIX. The Sandbox observes all system
calls made by the code, and compares them to the agency-defined policy. If any violations
occur, the Agency can shut down the suspicious agent.

The structure of the pattern is shown in Figure 2.20. The Agency depends on the
Sandbox to observe and control the Requester Agent’s activities, and the Sandbox
depends on the Agency to know the agency’s policies for agents sent to the Sandbox.

***

**Benefits:**

- Agents not authorised but valuable for the agency can be executed without compro-
mising its security.

- The agency can identify possible attacks (by observing the actions of the agents in
  the sandbox).

**Liabilities:**
Some computational resources of the agency might be diverted to non-useful actions, if non-useful agents are sandboxed.

The use of a sandbox introduces an extra layer of complexity.

Related patterns:

- N/A

Access Controller

... you are using Agent Authenticator to ensure the requester agent’s identity. Now you need to restrict access to the agency’s resources. Many different agents can exist in an agency, which require access to the agency’s resources in order to achieve their operational goals. However, they should can only access specific resources.

***

Agents belonging to an agency might try to access resources that they are not allowed to access.

Allowing this to happen might lead to serious problems such as the disclosure of private information, or the alteration of sensitive data. In addition, different security privileges
will be applied to different agents in the agency. The agency should take into account its security policy and consider each access request individually.

Therefore:

**Intercept all requests for the agency’s resources.**

The agency uses an Access Controller to restrict access to each of its resources. Thus, when a Requester Agent requests access to a resource, the request is dispatched to the Access Controller, which then checks the security policy, and determines whether the access request should be approved or rejected. Only if the request is approved, is the request forwarded to the corresponding Resource Manager.

The structure of the pattern is shown in Figure 2.21. The Requester Agent depends on the Resource Manager for the resource, and the Agency depends on the Access Controller for checking the request. The Access Controller, in turn, depends on the Agency for receiving the security policy and for forwarding the request, which it forwards to the Resource Manager in case the request is approved.

***

Benefits:

- The agency’s resources are used only by agents allowed to access them.
• Different policies can be used for accessing different resources.

Liabilities:

• There is a single point of attack. If the Access Controller is compromised, the system’s access control system fails.

Related patterns:

• N/A

Qualitative Evaluation of the Language

An important question that might be raised is how well the proposed language might prevent developers from building systems that might contain security holes?. To provide an answer to this question we have to evaluate how well our language follows the guiding principles for secure information systems design developed by Viega and McGraw[VM02]. In accordance with these principles, the different patterns of the language allow developers to break up the security of the system to different components (principle of compartmentalize), that are simple to develop and manage (principle of keeping the system simple). For example, the Agency Guard indicates there is only one point of access on the system, reducing considerably the design efforts required if more than one access points were present on the system. Moreover, the introduction of the patterns of the language on the development of a system ensures that in case of failure, the system will fail safely (principle of failing securely), since if one component fails, security is still achieved with the rest of the patterns. For example, if the Authenticator pattern fails, the Sanbox pattern ensures that any agent coming to the system will be running on a restricted environment without any privileges that might endanger the security of the system. To protect a system, the language proposes different levels of security including authentication, access control, and sandboxing, and as a result it promotes the principle of practicing defence in depth by avoiding a monolithic solution that would provide security only on one level of defence. The Access Controller pattern ensures that agents are allowed access only to resources they need and as a result it practices the principle of least privilege. In addition, the Authenticator pattern ensures that only authorised entities have access to specific information resulting in following the principle of promoting privacy. Finally, the language promotes the principle of community resources by employing patterns that have been derived from well-tested agent security solutions.

Although the principles cannot guarantee 100% security\(^2\), their authors estimate they cover about 90% of all potential problems by providing help in three different ways: (1) Help to prevent common errors during the development process, (2) Help to cope with unknown attacks, (3) Facilitate the understanding of security patterns and give security insight.

\(^2\)No design can actually guarantee 100% security.
2.3.4 Applying the Security Patterns

In this section we describe how our security patterns can be employed during the architectural design stage of the Tropos methodology. The integration of the patterns at this stage of the methodology will help identify additional actors to fulfill the security goals of the system without putting its security at risk.

During architectural design the system’s global architecture in terms of its subsystems (actors) and their interconnection via data and control flows is defined. From the point of view of security, an important step of architectural design is to identify the actors that take responsibility for achieving the system’s security goals. Security patterns can greatly help identify those actors without putting the security of the system at risk.

Although different developers might approach the application of the pattern language differently, depending on their experience, the following guidelines can be adopted:

1. **Identify the secure goals of the system.** During this step, the developer identifies all the secure goals of the system as derived during the late requirements analysis (modeled in 2.16). For instance, three are the main secure goals for the eSAP system (Figure 2.16): Ensure Data Availability, Ensure Data Integrity, and Ensure System Privacy.

2. **Identify the secure tasks for each of the secure goals.** This step involves the identification of the different secure tasks that correspond to the secure goals identified during the previous step. Consider as an example, the Ensure System Privacy secure goal of the eSAP system. This is achieved by the following secure tasks: Check Information Flow, Check Authentication, and Check Access Control.

3. **Identify the security challenge introduced by the secure tasks.** During this step, the security challenges with which the developer is faced, in order to satisfy the secure tasks are identified. Consider for instance, the Check Authentication and Check Access Control secure tasks identified in the previous step. To satisfy the former secure task, developers must make sure that all the communications and information exchanges of the eSAP system with external agents are authenticated, whereas to satisfy the latter, developers must make sure that every agent on the system only has access to designated resources. Therefore, for the above example, the security challenges can be summarised as:
   - Authenticate all the communications and information exchanges of the eSAP system
   - eSAP agents are only allowed to access designated resources

4. **Identify the patterns that correspond to the solution of the security challenges.** When the security challenges have been identified, the pattern language can be used to provide solutions to these challenges. For the above example, to provide a solution to the first challenge, the Agency Guard and the Agency Authenticator can
be employed, whereas to provide an answer to the second challenge, the Access Controller can be used. The Agency Guard pattern can be used to grant/deny access to the eSAP system as a whole, the Agent Authenticator pattern to provide authentication checks and the Access Controller pattern to perform access control checks.

In particular, consider the application of the patterns to the eSAP system with respect to the Obtain Updated Care Plan Information goal of the Older Person in Figures 2.14 and 2.15. The system needs to ensure that only entitled users can obtain access to this information. We start by applying the Agency Guard pattern, which restricts access to the agency to a single point. In the application of this pattern, the Older Person becomes the Requester Agent, the eSAP system corresponds to the Agency, and a new actor, the eSAP Guard, is introduced to assume the role of the Agency Guard. In Figure 2.22, we can see the result of applying this pattern. As shown, the pattern also introduces two new dependencies (Grant Access and Provide Security Policy).

Next we apply the Agent Authenticator pattern to ensure the identity of the Older Person agent (the Check Authentication subgoal of Ensure Data Privacy), and the Access Controller pattern in order to restrict the Older Person’s access to the agency’s resources, i.e., only to their own medical records (the Check Access Control subgoal of Ensure Data Privacy). The rationale for applying these patterns is that Agent Authenticator allows us to authenticate each agent that tries to access the system and using Access Controller restricts access to the resources of the system. The Sandbox pattern is not applicable, since the Older Person is a trusted user.

Application of these patterns leads to the introduction of more new actors and social dependencies between those actors and existing ones. The Authenticator fills the role of Agent Authenticator, and with it one new goal dependency (Authenticate Agent), and four resource dependencies (between Authenticator and eSAP Guard, and Older Person, respectively) are added to the model. Similarly, with the Access Controller two new goals (Check Security Policy, and Provide Security Policy), as well as two new task dependencies (involving the eSAP Agency, the eSAP Agency Guard, and the Care Plan Manager in the role of the Resource Manager) are introduced.

The use of the patterns of our pattern language helps developers to delegate the responsibilities of particular security goals to particular actors defined by the patterns. In addition, the developer knows the consequences that each pattern introduces to the eSAP system. For example, the application of the Agent Authenticator pattern means that during implementation only the agent authenticator must be checked for assurance, while the application of the Access Controller pattern implies that different policies could be used for accessing different resources.
Figure 2.22: Example of applying the security patterns. This diagram is the result of applying the patterns Agency Guard, Agent Authenticator, and Access Control in sequence.

2.3.5 Formalization of the Security Patterns

In this section we address the issue of the completeness of our language by formalizing properties of our patterns. The basic idea is to follow the “uses” links between patterns, which can be found in the Related Patterns sections, and record the problems addressed by each pattern, as well as the new problems they raise. From this we can either conclude that the application of our patterns helps establish security (that is, that all security problems raised are resolved), or that we need to add more patterns to our language in order to resolve the open problems. The properties of a pattern that we need to formalize are, therefore, problem, solution, and consequences (new problems raised). As Tropos is used to describe the solutions, we represent problems and solutions as Formal Tropos (FT) constraints [FLea04].

In general, each pattern makes three types of contributions: security solutions, and new security problems, as well as to non-security softgoals (e.g., complexity). The contributions to non-security softgoals could be used to compare alternative selections of patterns in terms of the quality of the overall solution (i.e., the combined result of applying the patterns). A solution that establishes security does not necessarily imply that it is the best solution in terms of other system qualities.

However, in the following, we restrict ourselves to security solutions and problems. We have also excluded liabilities, which are raised by the application of a pattern, but
which we have already resolved in their description, from the formalization. For reasons of space, we only present the formalization of patterns related to authentication and their relationships. For each pattern, we formalize the problem addressed, the solution, and the new problems introduced. The formalization of the problems appear where they are first raised, and are referenced in later patterns. (This approach also proved helpful in ensuring that the description of a problem does not use any of the new intentional elements introduced by the solution.)

FT is a specification language that offers all the standard mentalistic notions of Tropos and supplements them with a rich temporal specification language inspired by KAOS [DvLF91]. FT allows for the description of the dynamic aspects of Tropos models. More precisely, in FT we focus not only on the intentional elements themselves, but also on the circumstances in which they arise, and on the conditions that lead to their fulfillment. An FT specification describes the relevant elements (actors, goals, dependencies...) of a domain and the relationships among them. The description of each elements is structured in two layers. The outer layer is similar to a class declaration. It associates to the element a set of attributes that define its structure. The inner layer expresses constraints on the lifetime of the objects, given in a typed first-order linear-time temporal logic.

An excerpt of the outer layer of the FT specification for the Agency Guard pattern is given below.

Entity Resource
  Attribute owner: Agency
Actor AgencyGuard
Actor Agency
Actor RequesterAgent
  Attribute name: IDAgency
Goal Dependency ProvideSecurityPolicies
  Depender AgencyGuard
  Dependee Agency
  Mode achieve
Goal Dependency GrantDenyAccess
  Depender Agency
  Dependee AgencyGuard
  Mode achieve
Goal Dependency GainAccess
  Depender RequesterAgent
  Dependee AgencyGuard
  Mode achieve
Task Dependency RequestAccess
  Depender RequesterAgent
  Dependee AgencyGuard
Mode achieve

In the following an examples of constraints on the lifetime of class instances that define the inner layer of an FT specification.

Goal CheckRequest
   Actor AgentGuard
   Mode achieve
   Attribute constant resource : Resource
       constant rq : RequesterAgent
   Task dependency RequestAccess
   Depender RequesterAgent
   Dependee agencyGuard
   Mode achieve
   Attribute constant r : Resource
   Invariant r.owner = owner
   Fulfillment condition ∃ cr : checkRequest
      (cr.actor = self.dependee ∧ cr.resource = r ∧ cr.rq = self)

Agency Guard

The formalization of Agency Guard models the problem that requester agents can access the agency from multiple places via the GainAccess goal dependency. Problem P1 specifies that there is a way for a RequesterAgent to gain access to the agency by exploiting multiple GainAccess dependencies in which it participates. Solution S1 resolves this problem, as specified in the last clause of the assertion. Problem P2 also introduces the notion of an owner to be associated with any agent. In essence, the formalization of P2 states that ensuring agents can only access the agency through a single point does not also ensure that the agents are who they claim to be.

Problem

/* P1: A malicious agent can gain unauthorized access to the agency from multiple places, not all of which provide the same level of security. */

∃ ra : RequesterAgent (∃ ga1, ga2 : GainAccess (ga1.depender = ra ∧
   ga2.depender = ra ∧ ga1 ≠ ga2))

Solution

/* S1: Ensure that there is only a single point of access to the agency. */
∀ ra : RequesterAgent (∀ ga1,ga2: GainAccess (ga1.depender = ra ∧
  ga2.depender = ra) → (ga1.dependee = ga2.dependee))

New problems

/* P2: Agents can enter the agency by posing as another agent. */

∃ ar : RequestAccess (∃ ra : RequesterAgent (ar.depender = ra ∧
  ar.depender.name ≠ ra.name)

Agent Authenticator

The solution (S2) of Agent Authenticator resolves problem P2. It states that Requester-Agents signed with the private keys of their owners (the DigitalSignature element) can be authenticated via their public keys. However, this solution hinges on the fact that the agency knows the valid public key of the RequesterAgent’s owner. But this is generally not the case, as described by problem P3. In fact, a malicious agent may claim that its owner is ar.owner = ao1, whereas, it is ra.owner = ao2. The formalization introduces two new attributes: the key of a digital signature, and the privateKey attribute to be associated with agents and agent owners.

Problem

/* P2: Agents can enter the agency by posing as another agent. */

Solution

Actor RequesterAgent
  Attribute name: IDAgency
    key: DigitalSignature
    owner: AgentOwner

/* S2: Agents must prove their identity. Agents are authenticated via their own or their originating agency’s public keys. */

∀ ar : RequestAccess (∀ ra : RequesterAgent (ar.depender = ra ∧
  ∃ ao : AgentOwner (ra.owner = ao ∧ ra.key = ao.privateKey) →
  ar.depender.name = ra.name) /* ra knows ao’s private key */
New problems

/* P3: The agent’s public key may not be valid or certified. A malicious agent can exploit this by signing with its own private key. */

\[\exists \text{ar} : \text{AccessRequest} \ (\exists \text{ra} : \text{RequesterAgent} \ (\text{ar}.\text{dependee} = \text{ra} \ \land \\
\exists \text{ao1}, \text{ao2} : \text{AgentOwner} \ (\text{ar}.\text{owner} = \text{ao1} \ \land \ \exists \text{ds} : \text{DigitalSignature} \ (\text{ds}.\text{dependee} = \text{ra} \ \land \ \text{ds}.\text{key} = \text{ao2}.\text{privateKey} \ \land \\
\text{ao1} \neq \text{ao2} \ \land \ \text{ra}.\text{owner} = \text{ao2}))\]

It is important to note that problem P3 is only stated in terms of the concepts used in the Agent Authenticator pattern. We consider this problem outside the scope of this paper, although it presents an opportunity to add greater levels of detail to this pattern language. Need to decide whether to include P3 in the formalization. It demonstrates the benefits of the formalization: it identifies possible gaps in the language, but we do not address this gap.

TODO: Vet the formalization. Best to do this after we incorporate the updated diagrams (the formalization was prepared for the diagrams in the paper we had submitted to CAiSE.).

TODO: Add details on the FT model that would make the formalization easier to understand, ie include all actors and attributes referred to, as well as all new attributes added in the assertions.

Benefits of the Formalization

The formalization allows us to model how the application of a given pattern results in assertions being added to the model. We can now formally reason about the security problems resolved by a given security solution. For example, consider the assertion made by solution S2. It states that the owner of an AccessRequest must equal the owner of the RequesterAgent, if the request has been signed with the owners private key. Thus, the application of the Agent Authenticator pattern eliminates the possibility of one agent masquerading as another (formalized as problem P2).

Formalization also leads to a deeper understanding of the patterns and their interrelationships, and confidence in the quality of the pattern language.

2.3.6 Related Work

Our approach integrates two well known areas of software engineering research, agent-oriented software engineering and security patterns.
Agent-Oriented Software Engineering

Liu et al.[LYM02a] have presented work to identify security requirements using agent oriented concepts, whereas Yu and Cysneiros[YC02] provide an approach to model and reason about non-functional requirements (with emphasis on privacy and security). In his Non-functional Requirements (NFR) framework [CNYM00], Chung applies a process-oriented approach to represent security requirements as potentially conflicting or harmonious goals and he explains how they can be used during the development of software systems. In addition, Jürgens proposes UMLsec, an extension of the Unified Modelling Language (UML), to include modelling of security related features, such as confidentiality and access control[Jö01]. The concept of obstacles is used in the KAOS framework [DvLF91] to capture undesired properties of the system, and define and relate security requirements to other system requirements. Sindre and Opdahl define the concept of a miuse case to describe the security-related functions that a system should not allow[SO00a].

The main problem of these approaches is that they only provide solutions to specific problems. In other words, they consider security as a one-dimensional problem. For instance, the NFR framework assumes that all developers have some kind of security knowledge. UMLsec only supports the design phase, whereas the work of Liu et al. is focused on the requirements elicitation stage. By contrast, our approach considers all development stages from early requirements elicitation to the design of the system.

Security Patterns

The idea of developing a set of patterns or a pattern language for capturing proven security solutions is, by itself, not new. Yoder and Barcalow proposed a set of patterns that can be applied when adding security to an application[YB97a]. Lee Brown et al. proposed an Authenticator pattern, which performs the authentication of a requesting process before granting access to distributed objects[BF99]. Building on this work, Fernandez and Pan document a pattern language for security models[FP01a]. Finally, Schumacher applies the pattern approach to the security problem by proposing a set of patterns, which contribute to the overall process of security engineering[Sch03].

Although this review is by no means complete, most of the proposed security-related patterns and pattern languages have been developed from an object-oriented perspective. As stated by Fernandez and Pan, the intent of these patterns is to "specify the accepted models as object-oriented patterns"[FP01a]. However, it has been argued that there is no single paradigm or language for implementing patterns. Patterns can be integrated with any paradigm used for constructing software systems. We believe that the introduction of the agent-oriented paradigm has opened another important area for the use of patterns, as discussed next.

44
Merging these Areas

One of the main arguments\cite{BGG04a, Jen01} for the use of an agent-oriented, as opposed to an object-oriented approach, is that agent-oriented concepts for the decomposition of a system, such as goals, plans and actors, are more intuitive and easier to use than object-oriented concepts, such as data, behaviour and objects. In addition, the agent approach provides a paradigm for designing the types of complex distributed systems that require, among other things, a common abstraction for representing both human users (via agents that act on their behalf), as well as software entities that manage other subsystems or resources (agents as resource managers). Thus, the combination of patterns and agent orientation is likely to be very important, since the higher level of abstraction and the encapsulation of agents allows an easier identification and characterisation of reusable parts.

However, if patterns and pattern languages are to be integrated with agent-oriented approaches, it is necessary to develop them using agent-oriented concepts, such as intentionality, dependency, autonomy, sociality and identity\cite{Yu95d}. In doing so, we feel it is essential to describe the structure of a pattern not only in terms of the messages exchanged between the participating agents, but also in terms of their social dependencies and intentional attributes, such as goals and tasks.

In this way, we can achieve a complete understanding of the social and intentional dimensions of a pattern, two factors of great importance when developing information systems from an agent-oriented perspective. This view has been also argued by other researchers. For example, Deugo et al.\cite{DWK01} conclude that differences in the way agents and objects communicate and interact with their environment, the level of autonomy agents possess, and the fact that agents are often highly mobile motivate the separate notion of an agent pattern.

Thus, research on agent patterns has started to evolve and some catalogues of agent patterns have already been presented in the literature. Hayden et al.\cite{HCY99}, provide a catalogue of coordination patterns inherent in multi-agent architectures. The proposed patterns are grouped into four basic architectural styles: hierarchical, federated, peer-to-peer, and agent-pair. In the hierarchical style, a top-down control is imposed by agents in a supervisory or managerial role, while in the federated style, a main system provides overall coordination that the agents submit to. In the peer-to-peer style, individual agents are responsible for managing coordination and potential conflicts with others, while the agent-pair style describes one-on-one interactions. Kendall et al.\cite{KKPS98}, propose patterns for intelligent and mobile agents. In this work the authors argue that agent systems must have a strong foundation based on well defined patterns. Aridor and Lange\cite{AL98}, present a catalogue of design patterns for creating mobile agent applications. They divide their patterns into three classes: travelling, task and interaction.

Although these approaches are helpful and provide a first step for the integration of patterns and agent-oriented software engineering, they also demonstrate two important
limitations. Firstly, there is a lack of a framework to support the analysis of the require-
ments, and determine precisely the context within which a pattern can be applied. Sec-
ondly, and more specific to our topic, there is a lack of agent-oriented security patterns.
Our work provides a step towards overcoming these limitations by defining an agent-
oriented security pattern language, and by integrating this language with an agent-oriented
software engineering methodology.
2.4 Security Patterns for Agent Systems

The idea of developing a set of patterns or a pattern language for capturing proven security solutions is not new. Yoder and Barcalow [YB97b] proposed a set of patterns that can be applied when developing security for an application. F. Lee Brown et al. [BDdVF99] proposed a pattern, called Authenticator, that performs authentication of a requesting process before deciding access to distributed objects. Fernandez and Pan propose a pattern language for security models [FP01b].

Although this review is by no means complete, most of the proposed security related patterns and languages have been developed having object orientation in mind and they mainly neglect agent orientation concepts. As Fernandez and Pan state [FP01b] "Our intent is to specify the accepted models as object-oriented patterns". However, if patterns and pattern languages are to realise their potential in the development of agent-based systems, then it is necessary to develop patterns and pattern languages that are specifically tailored to the development of agent-based systems, using agent-oriented concepts. As Wooldridge et al state [JW01], "there is a fundamental mismatch between the concepts used by object-oriented developers (and indeed, by other mainstream software engineering paradigms) and the agent oriented view".

The proposed pattern language contains four patterns and it documents how an agency can be protected from malicious agents/agencies. The patterns of the language are categorised into two main categories: Patterns that deal with agency's access issues, such as authentication, authorisation, and access control, and patterns that deal with communication issues of the agency, such as secure communication with other agencies, and repudiation.

To model our pattern language we employ agent-oriented concepts used in the development of agent based systems. We feel this is necessary in order to make the pattern language applicable to agent developers. Agent orientation is based around the concept of an agent. According to Yu [Yu95a] an agent as a modelling construct demonstrates the following characteristics:

- **Intentionality.** An agent can be modelled in terms of its intentional properties, such as goals, tasks, resources, beliefs and capabilities, without having to know its specific actions in terms of processes and steps. Although such a high level abstraction does not provide a complete specification for the implementation of the system, it provides developers the ability to model the functional and non-functional requirements of the system and distinguish between different alternatives at an initial stage of the development.

- **Autonomy.** Agents are autonomous and can act independently. Because of the autonomy and independence agents are free to choose from a variety of different

---

3See http://www.securitypatterns.org for a complete review of security related patterns
actions to perform. Using the concept of a goal helps to model this kind of behaviour since a goal implies that they might be many ways of achieving it.

- **Sociality.** An agent most likely participates in relationships with other agents. In many traditional software engineering techniques, relationships are focused only on the exchange of data and intended functions. However, agent relationships are similar to human relationships and thus much more complex. Agent relationships involve conflicts amongst the relationships, multi-lateral relationships, and delegation of relationships.

- **Identity and boundary.** Agent orientation does not necessarily bounds the modelling concept of an (abstract) agent to that of a physical agent. Agents can be described across a range of physicality and abstractness. For example, social agents can often create new abstractions such as roles, and positions to help to define each others responsibilities and functionality.

We believe it is necessary to employ the above concepts when describing patterns for agents based systems. In doing so, we feel it is necessary to describe the structure of a pattern not only in terms of the collaborations and message exchange between the agents but also in terms of the social dependencies and intentional attributes, such as goals and tasks, of the agents involved in the pattern. This way we can achieve a complete understanding of the pattern’s social and intentional dimensions, two factors very important on agent-based systems.

To describe our patterns we are employing Tropos [CKM01b] methodology that adopts concepts from the $i^*$ modelling framework [Yu95a]. The main modelling concept is that of an actor. An actor has intentional properties (Intentionality) and is autonomous (Autonomy). Actors can be (social) agents (organisational, human or software), positions or roles (Identity and Boundary). Actors have social dependencies (Sociality) for defining the obligations of actors (dependees) to other actors (dependers). 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; task dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender. By depending on others, actors are able to achieve goals that will be very difficult or impossible to achieve on its own. To make the concepts more understandable, we consider a small example. In our example we consider three actors:

- **Agency.** An agency represents the environment in which an agent runs. At least one agency must be active on each host computer to enable it to execute agents.

- **Security Manager.** This actor represents an agent that is responsible for the security of the agency.

- **External Agent.** It is an agent that does not belong to the Agency that wishes to access some information of the agency.
Figure 2.23 represents the relationships between those actors in terms of their social dependencies. The main goal of the external agent is to access agency information. However, the agency will allow only authorised agents to access information. To fulfil the secure agency access goal the agency depends on the Security Manager. The security manager on the other hand, depends on the External Agents in order to obtain his access detail and be able to decide to allow or deny access to the agency information.

Figure 2.23: An example of representing social dependencies between different actors of a system.

2.4.1 Selected Patterns of the Language

**Name:** AGENCY GUARD (AG)

**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 access to agencies that are not allowed. A malicious agent that gains unauthorised access at the agency, it can disclose, alter or generally destroy data resided in the agency. 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 and thus deny services to authorised agents of the agency. The problem becomes greater if many “back-doors” are available in an agency, enabling potential 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 rather access should be granted based on the security policy of the agency.
Figure 2.24: The AGENCY GUARD Dependencies.

**Solution:** There must be only a single point of access to the agency. When an agent (Requester Agent) wishes to access resources of an Agency or move to this agency, its requests must forwarded through the AGENCY GUARD that is responsible to grant or deny 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. In traditional terms the concept of an AGENCY GUARD is referred to as the reference monitor [Amo94b].

**Social Dependencies:** A graphical representation involving the actors of the pattern and their dependencies is shown in Figure 2.24. The Agency depends on the AGENCY GUARD to grant/deny access to the agency. The AGENCY GUARD will grant/deny access according to the security policy of the agency, thus in order to fulfill this goal it depends on the agency to obtain the security policies.

**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 security policy changes (Not all the agents in the agency).
+ Only the guard must be tested for correct enforcement of the agency’s security policy.
+ Only one point of access to the agency, not many backdoors.
  - 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 [KVC96] and the EMBASSY patterns [KGM01]. 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 SANBOXING pattern can be used.

![Diagram of AGENT AUTHENTICATOR Dependencies](image)

**Figure 2.25: The AGENT AUTHENTICATOR Dependencies.**

**Name:** AGENT AUTHENTICATOR (AA)

**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.

**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 2.25. The AGENCY GUARD depends on the AGENT AUTHENTICATOR to authenticate the agent, and for such dependency the AGENT AUTHENTICATOR has to receive from AGENT GUARD the request (resource dependency - rectangle shape). The AGENT AUTHENTICATOR has to send the notification to the AGENCY GUARD when the agent is authenticated.

Consequences:

+ Authentication concerns are only dealt once. It is not necessary to make the agents of the system more complex by providing each one of those with an authentication mechanism
+ Ensures that an agent is authenticated before actually request a resource of 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 [LP02]. 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 [YB97b]. Moreover, applying the SANDBOX pattern can be used to restrict the set of resources available to the agent.

Name: SANDBOX

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 agency not trusted by the receiving 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 putting in danger its security.

**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 code from doing something it is not authorised to do. The code 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 code cannot do anything without first interacting with the operating system. Thus, SANBOX observes all system calls made by the code 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 2.26. The agency depends on the SANDBOX agent for observing and controlling the agent’s activities, and the SANDBOX depends on the Agency to know adopted policies.

**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).

![Diagram of ACCESS CONTROLER Dependencies](image)

Figure 2.27: The ACCESS CONTROLER Dependencies.

Name: ACCESS CONTROLER (AC)

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?

**Solution:** An ACCESS CONTROLLER exists in the agency. The ACCESS CONTROLLER controls access to each resource. Thus, when an agent requests access to a resource, the request is forwarded to the ACCESS CONTROLLER. The ACCESS CONTROLLER checks the security policy and determines whether the access request should be approved or rejected. If the access request is approved the ACCESS CONTROLLER forwards the request to the RESOURCE MANAGER.

**Social Dependencies:** The graphical representation of the pattern dependencies is shown in Figure 2.27. The Requester Agent depends on the Resource Manager for the resource, and the Agency depends on the ACCESS CONTROLLER for checking the request. ACCESS CONTROLLER depends on the Agency for receiving the security policies and for forwarding the request, that is forwarded to the Resource Manager in case it is approved.

**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 CONTROLLER pattern has been inspired by the ROLE-BASED ACCESS CONTROL pattern presented by Fernandez [FP01b]. 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.
2.5 Towards the Development of Secure Information Systems: Security Reference Diagrams and Security Attack Scenarios

In this section, we present the electronic Single Assessment Process (eSAP) system case study, first introduced [MPM02b] by one of the authors, which we will use throughout this section to illustrate our approach.

The eSAP case study involves the development of a health and social care information system for the effective care of older people in England [MPM02b]. Security is a very important factor in the development of the electronic Single Assessment Process, since security of personal health information is considered 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.

Moreover, health and social care professionals and older persons are worried that using such a system introduces risks for the privacy. Therefore, the 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 when developing the system.

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 neither 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.

2.5.1 Security Reference Diagram

Tropos [PBG+01a] is a software development methodology that employs modelling concepts such as actors, goals, soft goals, tasks, resources and intentional dependencies. Actors have strategic goals and intentions within the system or the organisation and represent (social) agents (organisational, human or software), roles or positions (represent a set of roles). A goal represents the strategic interests of an actor. In Tropos the concepts of hard goals (only goals hereafter) and soft goals are differentiated. Thus, soft goals have no clear definition or criteria for deciding whether they are satisfied or not [Yu95b]. Tasks represent a way of doing something and are usually executed in order to satisfy a goal. On the other hand, a resource represents a physical or an informational entity whereas a dependency between two actors indicates that one actor depends on another to accomplish
a goal, execute a task, or deliver a resource.

These concepts and the ability to cover the very early requirements analysis phase, allows for a deeper understanding of the environment, in which the system would operate, and the kind of interactions that occur between the system and any human users. Moreover, these concepts are suited to model security requirements, since these (security requirements) are usually expressed in natural language using notions such as agents and high level goals such as confidentiality and authentication [GMM03].

In previous work [MGM03c, MGM03e], we have introduced extra concepts to the Tropos methodology and we have also redefined existing concepts with security in mind, to enable Tropos to model security requirements during the software development process. However, the identification of security requirements in the very early phase of the software development is still a problematic activity. In order to help developer in their work we introduce the security reference diagrams.

The security reference diagram represents the relationships between security features, threats, protection objectives, and security mechanisms and its main purpose is to provide security novices with a valuable reference point when considering security issues during the development of information systems. Moreover, the security reference diagram aims to allow flexibility during the development stages of an information 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 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.

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 was decided 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 [Yu95b]. 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 a 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 work, 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 endanger the security features of the system. Since Tropos notation does not provide any related concept to model threats, a new notation has been introduced. Examples of threats are social engineering, password sniffing and eavesdropping attacks.

The notation of the above concepts is illustrated in Figure 2.28.

Figure 2.28: Notation used in 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 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 (-) (see Figure 2.29 as an example).

The process for constructing the security reference diagrams can be seen as a graph transformation process [AEH+99, BG02] that 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.

This process is non-deterministic due to the choice of a particular rule, at each step. To control this kind of non-determinism during the construction of the security reference diagram, priority rules have been assigned. These rules, introduced in [Mou03], are presented below in priority sequence.

**R1.** Introduce the security features to the diagram

**R2.** Introduce the security threats and associate them with the security features

**R3.** Introduce the protection objectives and associate them with the security features

**R4.** Introduce the security mechanisms and associate them with the protection objectives

**R5.** Decompose the security mechanisms to security sub-mechanisms

**Security reference diagram for the eSAP System**

From the discussion presented in Section 2, 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 2.29.

As shown in the diagram, each of those security features receives (according to the second rule (R.2) presented in the previous section) some negative contributions from different security threats.

Security threats to the electronic single assessment process (eSAP) are mainly the same as in any other medical system. According to Anderson [And01b] the main threat to medical privacy is social engineering [And01b]. 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 [And01b].

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.

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 forms 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.

On the other hand, the security features receive (according to the third rule (R.3) of the previous section) some positive contributions from different protection objectives. For instance, Privacy receives positive contributions from Authorisation and Cryptography. In turn, each of these protection objectives receives some positive contributions (according to the fourth rule (R.4) of the graph transformation rules) from different security mechanisms. For example, the Authorisation protection objective receives positive contributions from the Access Control, Authentication and Information Flow security mechanisms. Moreover, these security mechanisms can be decomposed (according to the fifth rule (R.5) of the graph transformation rules) to sub-mechanisms. For instance, the Authentication security mechanism is decomposed to Passwords, Digital Signatures and Biometrics.

As presented in [MGM03c, MGM03e], during the rest of the early requirements anal-
ysis, the late requirements analysis and the architectural design, the actors of the eSAP system along with their secure capabilities and the interactions between them are identified. As a result of this analysis [Mou03], the eSAP actor is decomposed, amongst others, to the following actors: Skills Manager, Assessment Evaluator, Integrity Verification Manager, Professional DB Manager, Authenticator, Actions Manager, Auditing Manager, Access Controller, and eSAP guard.

Having identified the actors of the system, the next step (during the architectural design phase) involves the application of the Security Attack Scenarios to allow developers to test the security of the developed system against possible security attacks.

2.5.2 Security Attack Scenarios

As mentioned in the Introduction, an important issue is to test how the system under development copes with any possible security attacks. According to the IEEE Standard Glossary of Software Engineering [iee83], 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 [RG99]. As a result, the applicability of those approaches conflict with some of the requirements, such as be clear and well guided, allow non-security specialists to consider security issues in the development process and to use the same concepts and notations [RG99]. Therefore, a technique, which is based on the use of scenarios and uses the same concepts and notations has been developed and integrated within the Tropos 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 [RG00]. In this research, scenarios are represented as enhanced Tropos diagrams and are used to test how the system copes in 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, we define a Security Attack Scenario (SAS) as an attack situation describing the actors of a software 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.
Thus, Security Attack Scenarios allow identifying the goals and the intentions of possible attackers, identifying through these a set of possible attacks to the system (test cases), and applying 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 (failed test cases) and this leads to the re-definition of the actors of the system and the addition of new secure capabilities to enable them to protect against those attacks.

A Security Attack Scenario involves a possible attacker, possible attack(s), the resources that are attacked, and the actors of the system related to the attack together with their secure capabilities. An attacker is depicted as an actor 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) that contain an ”attacks” tag, starting from one of the attackers goals and ending to the attacked resource (see for example Figure 2.30).

For the purpose of a Security Attack Scenario, a differentiation takes place between internal and external actors of the system. Internal actors represent the core actors of the system whereas external actors represent actors 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 actors 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 it is highly iterative and stages can be interchanged according to the perception of the developers.

**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.

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. For example, the threat ”social engineering” can introduce a goal ”perform social engineering” 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.

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

The next step in the creation of a security attack scenario involves the identification of the actors of the system that possess capabilities to prevent the identified, from the previous step, attacks. Therefore, the actors (internal and external) of the system related to the identified attack(s) are modelled. The secure capabilities, of each actor, 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.

**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 an effective means for document-based validation [KPW] 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 inconsistency between the scenarios and the models of the previous stages. Such an inspection involves the use of validation checklists.

It must be noticed that although inspections have been proposed by this research for the validation of the security attack scenarios, different techniques could be applied depending on the developers and the nature of the system. As an example, validation techniques to requirements specification are (apart from inspections) walkthroughs and prototyping [KPW].

**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. For each test case a precondition is necessary (the state of the system before the attack), an expected system reaction (how the system reacts in the attack), and also a discussion that forms the basis for the decision regarding 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 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.

**Security attack scenarios for the eSAP System**

As it can be seen from the security reference diagram, three are the main security features required by the eSAP system: privacy, integrity and availability. According to Stallings [Sta99b], the following categories of attacks can be identified that can endanger these 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 the privacy.

2. Modification, in which an unauthorised party not only gains access 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.

Due to lack of space, in this section we only present a security attack scenario related to a modification attack. 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 (Figure 2.30).

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 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 a Social Worker wishes to obtain an assessment evaluation [Mou03]. Three main test cases are identified: cryptographic attacks, data changing attacks and viruses attacks.

<table>
<thead>
<tr>
<th>Test Case 1: cryptographic attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition:</strong> 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.</td>
</tr>
<tr>
<td><strong>System expected security reaction:</strong> The eSAP system should be able to detect any kind of modification to the exchanged resources.</td>
</tr>
<tr>
<td><strong>Discussion:</strong> 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 [Sta99b]. 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, 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.</td>
</tr>
<tr>
<td><strong>Test Care Result:</strong> The system provides mechanisms to detect any modifications resulting from cryptographic attacks.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 2: changing data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition:</strong> The Attacker tries to change values of data stored in the eSAP system.</td>
</tr>
<tr>
<td><strong>System expected security reaction:</strong> The system should prevent attacks towards the unauthorised manipulation of its data.</td>
</tr>
<tr>
<td><strong>Discussion:</strong> 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 only access to limited 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.</td>
</tr>
<tr>
<td><strong>Test Care Result:</strong> The system provides mechanisms to protect against possible attacks aiming to change the data of the system.</td>
</tr>
</tbody>
</table>
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 Care Result: 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 actor of the eSAP system, should be able to monitor the system and take effective measurements against any possible viruses.

Figure 2.30: Modification Attack Scenario.
Integration of the proposed approach to the Tropos development process

Tropos supports four development stages, namely early and late requirements, architectural design, and detailed design [PBG+01a]. These stages have been refined to accommodate the proposed security extensions. 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). These security constraints are furthered analysed [Mou03] and secure goals and entities are introduced to the corresponding actors to satisfy them.

During the late requirements stage, security constraints are imposed to the system-to-be (by reference to the security reference diagram) and are analysed according to the techniques presented in [Mou03]. Moreover, security goals and entities necessary for the system to guarantee the security constraints are identified.

During the architectural design any possible security constraints and secure entities that new actors might introduce are analysed. Additionally, the architectural style of the software 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 [Mou03]. Furthermore, the actors of the system are identified along with their secure capabilities and the security attack scenarios are used to test the security of the system under development.

Finally, during the detailed design stage, the components identified in the previous development stages are designed. In particular, actor capabilities and interactions taking into account the security aspects are specified. The important consideration, from the security point of view, at this stage is to specify the components by taking into account their secure capabilities.
Chapter 3

Application of Security concepts.

3.1 Using a Security Requirements Engineering Methodology in Practice: the compliance with the Italian Data Protection Legislation

3.1.1 The Italian Data Protection Legislation

Many countries have recently promulgated a new privacy legislation spurred by increased concerns over data protection. Table 3.1 gives a brief history of European and Italian legislation about protection of personal data and privacy.

<table>
<thead>
<tr>
<th>European Legislation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regulation No 45/2001 of the European Parliament and the Council of 18 December 2000 on the protection of individuals with regard to the processing of personal data by the Community institutions and bodies and on the free movement of such data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Italian Legislation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legislative Decree No 196 of 30 June 2003 Italian Personal Data Protection Code.</td>
</tr>
<tr>
<td>Legislative Decree No 467 of 28 December 2001 concerning corrective and additional provisions with regard to the protection of personal data in accordance with Act No 127 of 24 March 2001.</td>
</tr>
<tr>
<td>Act No 325 of 3 November 2000 on provisions concerning the adoption of minimum security measures for personal data processing in accordance with Act No 6/3 of 31 December 1996.</td>
</tr>
<tr>
<td>Legislative Decree No 381 of 30 July 1999 concerning provisions with regard to personal data processing for historical, statistical and scientific research purposes.</td>
</tr>
<tr>
<td>Presidential Decree No 718 of 28 July 1999 Regulation on minimum security measures for personal data processing in accordance with Act No 6/3 of 31 December 1996 (it has been repealed and replaced by Legislative Decree No. 196/2003).</td>
</tr>
<tr>
<td>Act No 6/3 of 31 December 1996 on protection of individuals and other subjects with regard to the processing of personal data (it has been repealed and replaced by Legislative Decree No. 196/2003).</td>
</tr>
</tbody>
</table>

Table 3.1: Brief history of European and Italian data protection legislation
In Italy, data protection legislation is less than a decade old. Transposing the EC Directive 1995/46 into Italian law, the Italian Data Protection Act decreed that personal data are to be processed “by respecting the rights, fundamental freedoms and dignity of natural persons, in particular with regard to privacy and personal identity”. This goal was achieved by imposing to every data controller a set of obligations:

- identification of all entities involved in data processing with their roles and responsibilities;
- assurance that the purpose of data processing is fair, lawful and legitimate;
- implementation of minimal precautionary security measures to reduce risks on data disclosure were clearly defined with a later regulation enacted by Decree on July 28th, 1999.

Innovation and Technologies Department enacted the Directive on Computer and Telecommunications Security in Public Administration on January 16th, 2002. It was the first Directive of Italian Government that forces the entire public administration process to assess the security of their information systems and the start of the necessary activities to ensure their compliance to a minimal security basis. This minimal security base is defined by six main features: security policy, organization (roles and responsibilities), procedures, management and control, risk analysis and staff training. It required the adoption of a procedure for computer security incidents management and the creation of Computer Emergency Response Team (CERT). The requirements were close but not identical to the ISO standard 17799.

Later EU and Italian legislation systematized the norms on privacy and data protection. It confirmed and integrated:

- the definitions of personal data, sensitive data, and data processing,
- the definitions of all entities involved in data processing, their roles and responsibilities (controller, processor, operator, subject),
- the obligations relating to public and private data controllers with specific reference to the legitimate purpose of data processing and the adoption of minimal precautionary security measures to minimize the risks on data.

Skipping over specific ruling penalties and procedures, the law included a technical annex that regulates the implementation of minimal precautionary security measures as authentication and authorization system, antivirus, data backup and restore, and structure.

These measures had to be detailed into a “Documento Programmatico sulla Sicurezza” (DPS). The DPS is a security policy document for the management of all aspects of security concerning - organization, technology and procedures - explicitly imposed as an obligation to data controllers by the Data Protection Code. Every organization was supposed
to draw up, update yearly and obviously deploy a DPS. Table 3.3 shows an item-by-item comparison of the DPS enacted by the University of Trento and ISO-17799.

### 3.1.2 University of Trento: Information System & Organization

Personal data are processed within University for institutional purposes: education and research. The University has enforced the Data Protection Act through a Privacy Internal Regulation on January 14th, 2002 that transposed general regulations into its internal organization: it sets the responsibility line relating to personal data processing from data controller, the Chancellor, through data processors identified with Faculty Deans, Heads of Department and Central Directorate Managers down to data processing operators. Every data processor is responsible on behalf of the controller to accomplish the obligations relating to personal data processed within its own organization, supported by the ICT Directorate with regard to the adoption of the minimal precautionary security measures for electronic data processing.

Central Administrative Directorates (where the bulk of data processing is done) manage and coordinate all activities to support education and research. The Data Controller is identified with the Chancellor and all administrative executive directors are Data Processor within their own Directorate. Within the University, we have 10 Directorates: Chancellorship, General, Governance Relations, HR, Budget and Finance, Student Affairs and University Relations, ICT, Facilities Services, Library, Rovereto Administrative.

The Chief Executive Officer (CEO) has a special coordinating role within University on behalf of the Chancellor to accomplish all obligations related to personal data processing. The Chief Information Officer (CIO) is responsible for the adoption of the minimal and suitable precautionary security measures for electronic personal data processing.

The ICT Directorate manages the IT systems. The substructures in charge of Information Systems and Network, manage all central information services and network infrastructure whereas the local systems and services are managed by ICT local garrisons. Based on the University Privacy Internal Regulation, the CIO is responsible to draw up and to update the DPS and to implement the minimal and suitable security measures. Furthermore, he designates Database Security Operators and Network Security Operators within central structure and local garrisons.

Williams [Wil01] proposes a maturity model to establish rankings for security in an organization (Table 3.2). Matched against this scale, the University of Trento can be ranked between 3 and 4. In particular 4(a) is not yet enforced whereas 4(b) and 4(c) are (almost entirely) enforced.

---

1. The University has a subsidiary in another city.
2. Security awareness briefings are restricted to technical staff whereas non-technical staff only receive notifications in occasion of major virus and worm attacks. Intrusion testing is still amateurish.
3.1.3 Security-Aware Tropos

Tropos [BGG⁺04b] is an agent-oriented software development methodology, tailored to describe both the organization and the system. In Tropos, one can capture not only the what or the how, but also the why a piece of software is developed. This allows a more refined analysis of the system’s functional requirements, and also of the non-functional requirements such as security.

Here we use Security-Enhanced Tropos [GMMZ04b]. We have the concepts of actor, goal, soft goal, task, resource and social relationships for defining the obligations of actors to other actors. Actors have strategic goals and intentions within the system or the organization. A goal represents the strategic interests of an actor. A task specifies a particular course of action that produces a desired effect, and can be executed in order...
to satisfy a goal. A resource represents a physical or an informational entity. The relationships we have considered so far are functional dependency, ownership, provisioning, trust, and delegation of permission. A functional dependency between two actors means that the dependee will take responsibility for fulfilling the functional goal of a depender. The owner of a service has full authority concerning access and usage of his services, and he can also delegate this authority to other actors. Delegation marks a formal passage between the actors. In contrast, trust marks simply a social relationship that is not formalized by a “contract” between the actors: such as a digital credential or a signed piece of paper attributing permission.

Various activities contribute to the acquisition of a first requirement model, to its refinement into subsequent models:

**Actor modeling**, which consists of identifying and analyzing both the actors of the environment and the system’s actors and agents;

**Dependency modeling**, which consists of identifying actors which depend on one another for goal be achieved, plans to be performed, and resources to be furnished, and actors which are able to provide goal, plans, and resources.

**Trust modeling**, which consists of identifying actors which trust other actors for goal, plans, and resources, and actors which own goal, plans, and resources.

**Delegation modeling**, which consists of identifying actors which delegate to other actors the permission on goals, plans, and resources.

**Goal refinement**, which consists of refining requirements and eliciting new relations. This is standard in Goal-Oriented Methodologies [BGG+04b].

A graphical representation of the model obtained following the first four modeling activities is given through three different kinds of actor diagrams: functional dependency model, trust model, and trust management implementation. In these diagrams, actors are represented as circles; goals, tasks and resources are respectively represented as ovals, hexagons and rectangles.

Once the stakeholders and their goals and social relations have been identified, the analysis tries to enrich the model with more details. Goal refinement aims to analyze any goals of each actor, and is conducted from the perspective of the actor itself by using AND/OR decomposition. A graphical representation of goal refinement is given through goal diagrams. The outcome of this phase is a set of social relations among actors, defined incrementally by performing goal refinement on each goal, until all goals have been refined. Goal refinement builds goal hierarchies where lower goals are more specific and are motivated by goals higher in the hierarchy.
3.1.4 Modelling Actors

The first activity in the early requirements phase is actors’ modeling. This phase consists of identifying and analyzing the application domain stakeholders and their intentions as social actor which want to achieve goals.

In our example we can start by informally listing some of them. The following definitions\textsuperscript{3} apply and shall be used in this chapter:

**Data Controller** is the natural or legal person which determines the purposes and means of the processing of personal data. In the University, the data controller is identified with Chancellor (as the post-holder is also the legal representative of the University).

**Data Processor** is a natural or legal person which monitors personal data processing on behalf of the controller. In the University, based on the enacted regulations, data processors are identified with:

- Faculty Deans;
- Heads of Department;
- Central Directorate Managers, and in particular with:
  - Chief Executive Officer (CEO);
  - Chief Information Officer (CIO).

**Data Processing Operator** is the human appointed by the data controller or processor to perform the operations related to the data processing or to manage and maintain the information systems and services. At University of Trento, these are identified with:

- Personal Data Processing Operator;
- Database Security Operator;

**Data Subject** is the natural or legal person to whom the personal data are related. In the Secure Tropos terminology, this is the legitimate owner of the data.

**CERT (Computer Emergency Response Team)** is composed by:

- the staff of ATI Network that manages the network infrastructure and services of the University;
- the Information Security Office Manager;
- the CIO.

\textsuperscript{3}See Article 2 “Definitions” of EU Directive 95/46/EC.
To be more precise CERT includes a member in charge of security issues for every major ICT service center in the University.

In the underlying formal model based on datalog instances of actors are represented as constants satisfying atomic predicates for actors’ types (e.g. being Chancellor) and binary predicates are used to link agents and goals.

### 3.1.5 Modelling Dependencies and Delegation

The analysis proceeds introducing the functional dependencies and the delegation of permission between actors and the consequent integrated security and functional requirements. Figure 3.1(a) and Figure 3.1(b) show the functional dependency model and the trust management implementation. We use delegation of permission \((Dp)\) to model the actual transfer of rights in some form (e.g. a digital certificate, a signed paper, etc.), and \(Df\) for functional dependency.

In the functional dependency model, **Chancellor** is associated with a single relevant goal: *guarantee correct data processing execution*, while **CEO** has an associated goal *compliance with legal requirements*. Along similar lines, **Data Processor** and **Data Processing Operator** want to *comply with internal orders and regulation*, while **CIO**, wants to *guarantee law enforcement*. Finally, the diagram includes some functional dependencies: **Data Subject** depends on **Chancellor** for *privacy protection* goal; **Chancellor** depends on **Data Processor** and **Data Processing Operator** to *perform data processing*; and, in turn, **Data Processor** depends on **Data Processing Operator** for it.

---

**Figure 3.1: Actor Diagrams**

(a) Functional Dependency Model

(b) Trust Management Implementation
In the trust management implementation, following the current practice Chancellor delegates permissions to perform data processing to Data Processor and Data Processing Operator. In turn, Data Processor delegates permissions to perform data processing to Data Processing Operator.

At this stage, the analysis already reveals a number of pitfalls in the actual document template provided by the ministry’s agency. The most notable one is the absolute absence of functional dependencies between the Chancellor and the CEO, who is actually the one who runs the administration. Such functional dependency is present in the Universities statutes, but not here (an apparently unrelated document).

Another missing part in the trust management implementation is the delegation of permission from the data subject. This can be also automatically spotted with the techniques developed in [GMMZ04b]. Somehow paradoxically (for a document template enacted in fulfillment of a Data Protection Act) the process of acquisition of data (and the relative authorization) is neither mentioned nor foreseen. In practice this gap is solved by the University by a blanket authorization: in all the paper or electronic data collection steps a signature is required to authorize the processing of data in compliance with the privacy legislation.

### 3.1.6 Goal Refinement

A first example of the goal refinement is given by the goal diagram depicted in Figure 3.2 for the Chancellor. The goal guarantee correct data processing execution is decomposed into distribute data processing and determine executive orders. We call this a “AND-
decomposition”. The goal *distribute data processing* is decomposed (OR-decomposition) into two subgoals: *outsourcing* and *distribute to internal staff*.

The security requirements of an organization outsourcing the management and control of all or some of its information system is addressed in a contract agreed between the parties. For example, the contract should address: how the legal requirements are to be met, e.g. data protection legislation; what arrangements will be in place to ensure that all parties involved in the outsourcer, including subcontractors, are aware of their security responsibilities; how the integrity and confidentiality of the organization’s business assets are to be maintained and tested; etc. In a nutshell the contract should say that the goal *guarantee correct data processing execution* is also fulfilled by the service supplier. The contract should allow the security requirements and procedures to be expanded in a security management plan to be agreed between the two parties. Following these requirements, the goal *outsourcing* is AND-decomposed into *identify data controller*, *identify responsibilities and tasks*, and *expect declaration of security compliance*.

The other hand, the goal *distribute to internal staff* is decomposed into *distribute responsibilities* and *provide data to other offices of the university and to press*. Distribute responsibilities consists into *define responsibilities for data processor* and *appoint data processor*. Since security roles and responsibilities should include implementing or maintaining security policy as well as any specific responsibilities for the protection of particular assets, or for the execution of particular security processes or activities, the goal *determine executive orders* is AND-decomposed into five subgoals: *data processing objectives*, *data processing procedures*, *choice of processing instruments and tools*, *security profile*, and *manage internal directives* for which Chancellor depends on CEO. Note here the gap: everything is “formally” decided by Chancellor and only the final executive regulations are delegated to the CEO. Only in theory objectives, procedures, processing instruments and security profile are defined by Chancellor, whereas they are just enacted by him.

A second example, in Figure 3.3, shows the goal analysis for CIO, relative to the goal *guarantee law enforcement*. This goal is decomposed into *fulfill administrative and technical duties* and *manage security measures*. The goal *fulfill administrative and technical duties* is decomposed into three goals: *manage user access profile* for which Data Processor depends on CIO, *check activities’ evolvement*, and *census data processing* for which CIO depends on Data Processor. The goal *manage user access profile* is decomposed into *create user access profile* and *guarantee authenticate connections*. The goal *create user access profile* is decomposed into *update authorization database*, *generate ID*, *generate and retrieve password*, and *communicate user access profile* for which Data Processing Operator depends on CIO. The goal *manage security measures* is decomposed into *define security measures*, *monitor security measures*, *verify security measures*, and *convey security measures* for which Data Processor depends on CIO. Essentially this map the formal

---

4The procedure also includes some fuzzy steps on something that is a security anathema (helping users who forgot their password) but a fairly frequent problem.
requirements that a policy document should be approved by management, published and communicated, as appropriate, to all employees.

The goal diagram in Figure 3.4 shows the trust management implementation for Chancellor with respect to goal guarantee correct data processing execution. In particular, it points out that Supplier delegates a signed declaration of security compliance to Chancellor where Supplier engages in honoring and enforcing the undertaken responsibilities. This map the formal requirements that the University has security policies that requires
adherence to several necessary precautions in order to maintain privacy protection in behalf of Data Subject. Further, Chancellor delegates mail within instructions to Data Processor and executive orders list to CEO.

Figure 3.5 shows the trust management implementation for CIO. The diagram displays that Data Processor delegates data processing list to CIO for census. Further, CIO delegates ID, password and user access profile to Data Processing Operator.

The model has been further refined down to the the various offices and members of staff until it could be matched one-one with the actual DPS. Next, we present other diagrams for the some actors involved in the system. Figures 3.6 and 3.7 show, respectively, functional dependency model and trust management implementation for Data Processor relative to the goal comply with internal orders and regulation. Figures 3.8 and 3.9 show, respectively, the goal refinement of the functional dependency model and the trust management implementation for Data Processing Operator, relative to the goal comply with internal orders and regulation, and for Database Security Operator, relative to the goal manage and maintain ICT instruments and tools.

The goal diagrams in Figure 3.10(a) and 3.10(b) show, respectively, functional dependency model and trust management implementation for Data Subject. The functional dependency model reveals that Data Subject depends on Chancellor to get informative, and that Chancellor depends on Data Subject for the consensus needed for performing data processing. The trust implementation model displays that Chancellor delegate informative to Data Subject, and Data Subject delegate consensus to Chancellor.

Figure 3.11 shows the goal refinement for CEO, relative to the goal compliance with legal requirements. The goal diagrams in Figure 3.12 shows the functional dependency model for CERT, relative to the goal co-ordinate response activities for security incidents,
and for Information Security Office Manager, relative to the goal manage information security and privacy matters. The trust implementation model for CEO and CERT is not shown since it is not defined in the DPS.

### 3.1.7 Adequacy and Analysis of the Model

The primitives suggested for Secure Tropos were sufficient to cope with the complexity of a real ISO-17799-like case study and the methodology allowed to pinpoint many issues.

For example, the first observation is that a trust model is not considered in the required procedures and documents. Trust relations are implicitly defined in the employment contract that actors draw up with the University. In absence of such model, some of the properties proposed in [GMMZ04b] cannot be verified since trust is at the base of such framework. Note also that, in according with the Code, data subjects own their personal data. In [GMMZ04b], we suggest to check if employees who are entitled to access to Personal data, have previously gotten the permission from data subjects for them. In above models, this is not verified since there is not delegation from data subjects to employees for personal data. Essentially we only have a blanket authorization.

Further, DPS defines only objectives and responsibilities for the entities involved into the organization, but does not identify who is really able to provide services. This entails that some relations among entities could miss. For example, looking at Figure 3.3 and 3.5, the CIO has the responsibilities to manage user access profile. In practice, he delegates...
the execution of this goal to an employee of the ICT Directorate that generates IDs and passwords, and then delegates them to data processing operators. Consequently, it is not possible capture requirements of availability unless an explicit model of the functional requirements is also given. For instance, we cannot verify whether data subjects delegate their personal data only to someone that is able to provide the requested service. This clashes with privacy principles and, specifically, with the notion of “limited collection”: the collection of personal information should be limited to the minimum necessary for accomplishing the specified service.

Notice that this is not a problem of the University of Trento, but rather of the entire security assessment procedure in the state of the art: unless the ISO-17799 policy (or its equivalent DPS) is matched by a description of the functional goals of the organization it is not possible to conclude whether access is fair or respect least privileges principles. The same problem affects EPAL proposals [BKBS04, BPS03] and other privacy proposals in the literature [AKSX02, AKSX04, CLMR02, KSW02].

The most painful (and so far not formally analyzed part) is the treatment of manual non-ICT procedures. This difficulty steams from two main sources. The first one is that non-ICT procedures are often not completely formalized since there is no need for “programming” and “debugging” a human. This does not means that offices do not follow standard procedures but rather that these procedures are somehow “embedded” in the or-

Figure 3.7: Trust Management Implementation for Data Processor
Data Processing Operator

comply with internal orders and regulations

manage data access

password protection

change password

communicate password change

comply with security regulations for data protection

modify password every 6 months

guard secret the password

change password every 6 months

organize or the “office distributed knowledge”. In absence of fully formalized functional procedures it is difficult to define the corresponding authorization and trust management procedures.

3.1.8 Related Case Studies and Conclusions

The last years have seen an increasing awareness that security and privacy play a key role in system development and deployment. This awareness has been matched by a number of research proposals on incorporating security and privacy considerations into the mainstream requirement and software engineering methodologies. Yet, only few papers describe complex case studies.

Becker et al. [BS04] use Cassandra to model and analyze an access control policy for a national electronic health record system. The background of this case study is the British National Health Service’s current plan to develop an electronic data spine that will contain medical data for all patients in England. The proposed policies contain a total of 310 rules and define 58 parameterized roles.

In [AE04], Antòn et al. introduces a privacy goal taxonomy and reports the analysis of 23 Internet privacy policies for companies in three health care industries: pharmaceutical, health insurance and on-line drugstores. The identified goals are used to discover inner
A study of the certification of information security management systems based on internal conflicts within privacy policies and conflicts with the corresponding websites and their manner of manage customers’ personal data.
specifications promulgated by Taiwan’s Ministry of Economic Affairs is proposed in [FFL03]. In particular, this work shows the ability of Taiwan’s information security management systems to meet the requirements proposed in international standards. In [FLF04], authors analyze the knowledge and skills required for auditing the certification procedures for asset, threat, and vulnerability. They recognize that reducing risks is the target of information security management system protection mechanism. Thus, risk assessment is need to analyze the threats to and vulnerabilities of information systems and the potential impact of harm that the loss of confidentiality, integrity, or availability would have on an agency’s operations and assets.

In this chapter we have shown the Secure Tropos methodology at work on a real-life comprehensive case study encompassing on ISO-17799 security management policy. The proposed constructs and methodology were up the challenge and revealed a number of pitfalls, especially when the formal analysis techniques were applied.

Future work is in the full automated analysis of the policy at the level of individual staff members processing data.

### 3.1.9 Comparing ISO-17799 and DPS 3.0
Figure 3.12: Functional Dependency Model for CERT
Table 3.3: Comparing ISO-17799 and DPS 3.0
Table 3.3: Comparing ISO-17799 and DPS 3.0
11 SYSTEMS DEVELOPMENT AND MAINTENANCE
11.1 Security requirements analysis and specification
11.1.1 Security requirements analysis and specification
11.2 Security in application systems
11.2.1 Input data validation
11.2.2 Control of internal processing
11.2.3 Message authentication
11.2.4 Output data validation
11.3 Cryptographic controls
11.3.1 Policy on the use of cryptographic controls
11.3.2 Encryption
11.3.3 Digital signatures
11.3.4 Non-repudiation services
11.3.5 Key management
11.4 Security of system files
11.4.1 Control of operational software
11.4.2 Protection of system test data
11.4.3 Access control to program source library
11.5 Security in development and support processes
11.5.1 Change control procedures
11.5.2 Technical review of operating system changes
11.5.3 Restrictions on changes to software packages
11.5.4 Counter channels and Trojan code
11.5.5 Outsourced software development

12 Business Continuity Management
12.1 Aspects of business continuity management
12.1.1 Business continuity management process
12.1.2 Business continuity and impact analysis
12.1.3 Writing and implementing continuity plans
12.1.4 Business continuity planning framework
12.1.5 Testing, maintaining and re-assessing business continuity plans

13 Compliance
13.1 Compliance with legal requirements
13.1.1 Identification of applicable legislation
13.1.2 Intellectual property rights (IPR)
13.1.3 Safeguarding of organizational records
13.1.4 Data protection and privacy of personal information
13.1.5 Prevention of misuse of information processing facilities
13.1.6 Regulation of cryptographic controls
13.1.7 Collection of evidence
13.2 Reviews of security policy and technical compliance
13.2.1 Compliance with security policy
13.2.2 Technical compliance checking
13.3 System audit considerations
13.3.1 System audit controls
13.3.2 Protection of system audit tools

Table 3.3: Comparing ISO-17799 and DPS 3.0

<table>
<thead>
<tr>
<th>ISO-17799</th>
<th>DPS 3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1.1</td>
<td>6.2</td>
</tr>
<tr>
<td>11.2.1</td>
<td>6.3</td>
</tr>
<tr>
<td>11.3.1</td>
<td>6.4</td>
</tr>
<tr>
<td>11.4.1</td>
<td>6.5</td>
</tr>
<tr>
<td>11.5.1</td>
<td>6.6</td>
</tr>
</tbody>
</table>

87
3.2 Using Tropos Methodology to Model an Integrated Health Assessment System

3.2.1 The single assessment process case

National policy in England is to promote the Single Assessment Process (SAP), an integrated assessment of health and social care needs of older people. The Single Assessment Process aims to create closer working for providing primary health and social care for older people and other groups. Ultimately, this might lead to the development of Care Trusts, a single local organization for delivering health and social care. The development of integrated health and social care information systems will support closer working and facilitate the development of Care Trusts. In the SAP setting, different health care professionals, such as general practitioners, nurses and social workers, must cooperate together in order to provide patients with appropriate care.

With closer working, professionals will work in teams that will be responsible for the health and social care of the older person. Each team will demonstrate the following characteristics.

- Each team consists of many different professionals.
- Professionals cooperate between them.
- Professionals share information between them.
- Each professional has some expertise.
- Teams will promote person-centred care.

With the single assessment process, a common language of assessment will be used to support information sharing, and the potential to aggregate information to describe the health and social care needs of the local population of older people, which is an extremely useful tool for planning services and monitoring trends in needs and outcomes.

The single assessment process will also provide the older person and their carer with a personal copy of their care plan to support person-centred care. The single assessment process will work in three main stages. The contact, the assessment and the follow-up action. During the first stage contact assessment will provide basic information. In the second phase an overall assessment using a validated assessment instrument, such as Easy-Care [Phi97], will take place. The third stage will provide older people with more care in particular problems (might be different problems for each individual such as house and loneliness problems) and with more detailed assessment if appropriate. The selection of the problems is determined by the results obtained from the Easy-Care assessment instrument.
Computerising this process will help to automate some of the administration tasks, such as the appointments set up between the health and social care professionals, and the management of the health and social care teams, and thus leaving the professionals with more time for the actual care of the older person. Furthermore, it will help older persons to be actively involved in their health and social care, since they will have access to the system.

### 3.2.2 Applying Tropos methodology

Tropos is a methodology, for building agent-oriented software systems, tailored to describe both the organisational environment of a system and the system itself, employing the same concepts through out the development stages. Tropos adopts the i* modelling framework [Yu95c], which uses the concepts of actors, who can be (social) agents (organisational, human or software), positions or roles, goals and social dependencies (such as soft goals, tasks, and resources) for defining the obligations of actors (dependees) to other actors (dependers). This means the system (as well as its environment) can been seen as a set of actors, who depends on other actors to help them fulfil their goals. Tropos covers four phases of software development:

**Early Requirements**, concerned with the understanding of a problem by studying an existing organisational setting; the output of this phase is an organisational model, which includes relevant actors and their respective dependencies;

**Late requirements**, where the system-to-be is described within its operational environment, along with relevant functions and qualities; this description models the system as a (small) number of actors which have a number of dependencies with actors in their environment; these dependencies define the system’s functional and non-functional requirements;

**Architectural design**, where the system’s global architecture is defined in terms of subsystems, interconnected through data and control flows; within the framework, subsystems are represented as actors and data/control interconnections are represented as (system) actor dependencies;

**Detailed design**, where each architectural component is defined in further detail in terms of inputs, outputs, control, and other relevant information. Tropos is using elements of AUML [BMO01b] to complement the features of i*.

**Early Requirements Analysis**

As it was mentioned above, during the early requirements analysis the analysis engineer models the goals and the dependencies between the stakeholders (actors). For this purpose, Tropos introduces actor diagrams. In such a diagram each node represents an actor,
and the links between the different actors indicate that one depends on the other to accomplish some goals. The electronic Single Assessment Process project (eSAP) includes, among the others, the following stakeholders:

- Older Person (OP) is the patient that wishes to receive appropriate health and social care.
- Department of Health (DoH) is the government department that must provide older people with health and social care.
- R&D Agencies are agencies that wish to obtain information about older people to help them in their research.

These actors along with their goals are shown in Figure 3.13. The Older Person actor has as a main goal to receive appropriate health and social care and as a soft goal to Maintain Good health. The Older Person depends on the DoH to accomplish their goal and soft goal. On the other hand, DoH has as a main goal to Provide Better Health and Social Care to Elderly. The R&D Agency has an associated goal to Obtain Patient Information For Research.

When the stakeholders, their goals and the dependencies between them have been identified, the next step of this phase is to analyse in more depth each goal relative to the stakeholder who is responsible for its fulfilment. In doing so, Tropos adopts i* rationale diagrams for analysing the actors’ goals through a means-end analysis. Each rationale diagram is presented as a balloon within which the goals and the dependencies are analysed. Such an analysis from the prospective of the Older Person is shown in Figure 3.14.

As it was mentioned, the Older Person actor has as a goal to receive appropriate care and as a soft goal to maintain good health. The receive appropriate care goal is fulfilled by the tasks Undertake Assessment, Follow Care Plan, Get Info about Care Plan, Obtain Medical Info, and Have Appointment with Professionals. To perform the last three tasks
the Older Person must use the eSAP system, so the last three tasks are decomposed into the goal Use eSAP System. In addition, the Maintain Good Health soft goal depends on the Follow Care Plan and Undertake Assessment tasks to be fulfilled. Furthermore, the Older Person depends on the DoH to make the Technology Infrastructure Available, and also to make the eSAP System Available and Easy-to-Use.

Figure 3.14: Means-end analysis for Older Person.

Another important stakeholder of the system is the DoH. The rationale diagram for the DoH is shown in Figure 3.15. The main goal of the Department of Health is to Provide Better health and Social Care to Elderly. To accomplish this goal, DoH defines a sub goal to Make Care Person-Centred. The later is essential for the DoH to fulfil its main goal since the Older Person is the most important participant of the whole procedure, since they know better than anyone their difficulties and when they need health and social care. Thus, the Make Care Person-Centred goal is further decomposed into the two sub-goals Promote SAP and Involve Elderly in their Care. The later sub-goal depends on the task Provide Guidelines for the Older People to be fulfilled. The Promote SAP sub-goal is decomposed into the Computerise SAP goal and the Provide Guidelines for the SAP task. The later task is decomposed into two sub-tasks: Provide Guidelines for the Different Health and Social Care Professionals and Provide Guidelines for the Care-
Teams. The first sub-task is further decomposed into four sub-tasks: Provide Guidelines for GPs, Provide Guidelines for Nurses, Provide Guidelines for Other Professionals and Provide Guidelines for Social Workers. In addition, to fulfil the Provide Guidelines for the Care-Teams goal each locality must comply with the proposed guidelines. Provide Efficient Care is another important goal of the DoH. To accomplish this goal the sub goal Computerise SAP has been identified. Computerising the SAP will help health and social care professionals to automate some procedures required while caring of the older person and thus help to Provide Efficient Care. To accomplish the Computerise eSAP sub goal, Technology Infrastructure must be provided and the eSAP System must be available. The goal Build eSAP is motivated by these two goals since it has no sub-goals.

![Means-end analysis for Department of Health (DoH).](image)

**Late Requirements Analysis**

During the late requirements analysis the system-to-be (the eSAP system in our case) 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 functional and non-functional requirements for the system-to-be.

For the eSAP system the following, amongst others, three requirements have been defined:

1. The professionals must be able to customize their software agents through an easy-
to-use interface.

2. The system must be developed with mobility in mind since many of the professionals will use it whilst in the older person’s house.

3. The system must be secure.

The eSAP system is introduced as another actor, as shown in Figure 3.16. The DoH depends on the eSAP to Provide Efficient Care, and also to Make Care Person-Centred, in order to fulfil its main goal, which is to Provide Better Health and Social Care. In our example we concentrate on the analysis of the goals Provide Efficient Care, Make Care Person-Centred, and Usable eSAP System.

The goal Provide Efficient Care aims to allow health and social care professionals more time for the actual health and social care of the older person. Plenty of time is currently spent from the health and social care professionals for administrating procedures, such as appointments set up, and communication with colleagues. eSAP will minimise the time spend on these procedures by automating most of the tasks and providing services, such as access to medical libraries, and medical records. Thus, the Provide Efficient Care goal is decomposed into the Provide Services sub-goal, which is further decomposed into four sub-goals, Access to Medical Libraries, Access to Medical Records, Schedule Appointments and Access to Care Plans. The later goal is further decomposed into the sub-goal care plan info, which is decomposed into four further sub-goals Future Care Plan Appointments, Professionals Involved, Previous Assessments, and Future Actions.

Another important goal of the eSAP system is to promote person-centred care. To fulfil this goal, eSAP must Provide Facilities to Older People. The later is decomposed into two further goals Access to Care Plan and Access to Medical Info. The soft goal Usable eSAP System has 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 to be easy-to-use, and the latter contributes positively because the system must have a User Friendly Interface to be easy-to-use. 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 make it more difficult to be used.

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 for the system to be secure, and the latter because communications must be secured for the eSAP system to 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 be secured.

Figure 3.16: Means-end analysis for eSAP System.

When the system goals and soft goals have been identified, new actors and sub-actors are introduced. Each of the new actors takes the responsibility to fulfil one or more goals of the system. Figure 3.16 shows a partial decomposition of the actors and sub-actors of the eSAP system, along with their dependencies with respect to the eSAP system.

The eSAP system depends on the Medical Library Manager to provide Access to Medical Libraries, on the Medical Record Manager to provide Access to Medical Records, on the Appointments Manager to Schedule Appointments, and on the Care Plan Manager to provide Access to Care Plans. The Care Plan Manager depends on the Care Plan Appointments Manager to Access Care Plan Appointments, on the Professionals Manager to provide information about the professionals involved in the care plan, 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 eSAP depends on Security Manager to fulfil the Secure eSAP System goal. The Security Manager depends on the Authorisation Manager to Authorise Access to the System, on the Confidentiality Manager to assure the confidentiality of the data of the system, and on the Integrity Manager
to assure the integrity of the data.

Figure 3.17: Sub-Actors Decomposition for the eSAP System.

**Architectural Design**

The architectural design includes the following four steps:

- Addition of new actors, in which new actors are added to make the system interact with the external actors as well as to contribute positively to the fulfilment of some non-functional requirements. Tropos introduces the extended actor diagram in which the new actors and their dependences with the other actors are presented.

- Actor decomposition, in which each actor is described in detail with respect to its goals and tasks.

- Capabilities identification, in which the capabilities needed by the actors to fulfil their goals and tasks are identified, by analysing the extended actor diagram. Each dependency relationship can give place to one or more capabilities triggered by external events.

- Agents assignment, in which a set of agent types is defined assigning to each agent one or more different capabilities.
In the eSAP system, the software agents will act on behalf of professionals. Each professional will have his/her “own” software agent, which will be customised according to his/her needs. The agent will have enough information about the professional, such as personal information and professional commitments, and it will be intelligent enough (capable of analysing the information and take decisions) that will enable it to act on his/her behalf, and also negotiate for the interest of the professional.

From the above we decided for the following architectural choices:

1. The system will consists of software agents as well as human professionals.
2. Each professional will have his/her software agent.
3. Professionals will be able to customize the software agent according to their needs.
4. The software agent will be capable of analysing information and take decisions. Also, it will have information about the professional (personal and professional) that will be able to act on his/her behalf.
5. Software Agents in the system will be able to communicate between themselves as well as with the human professionals.

Figure 3.18 shows the extended actor diagram with respect to the Authorisation Manager. The Authorisation Manager is responsible for authorising access to the Older Person (O.P) when trying to perform the Get Info About Care Plan task. When authorisation is granted the O.P interacts with the Care Plan Manager in order to obtain available Care Plan Information. Then depending on the kind of information the O.P. wishes to access, he/she interacts with the appropriate manager. Again the Authorisation Manager checks for authorisation permissions and grants access to the Care Plan Data Manager, which manages a repository for the care plan information. To represent the interactions of the health and social care professionals within the system we have introduced the Professional actor. The Professional interacts with the Authorisation Manager to gain the authorisation to interact with the Medical Library, Medical Records, and Appointment managers. In addition, the Authorisation Manager interacts with the R&D Agency actor to grant authorisation to the Care Plan Data Manager.

The second step of the architectural design is the decomposition of actors in sub-actors aiming to expand in details each actor with respect to its goals and tasks. Figure 3.19 shows the decomposition of the Authorisation Manager Actor with respect to the get information about care plan task. The Authorisation Manager is decomposed in two sub-actors: the Authorisation Granter and the Authorisation Checker. The former is responsible for checking the users’ authorisation details and grant access to services, while the latter is responsible for checking if the user has access and in what services. The Authorisation Granter depends on the Authorisation Checker to obtain the Authorisation Privileges of the user. The Authorisation Checker interacts with the Authorisation Data in order to obtain information about the Authorisation Status of the user.
The next step of the architectural design is the capabilities identification, 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.

The last step of the architectural design is the agents’ assignment. During this step each agent is assigned one or more different capabilities identified in the previous step. Table 3.4 illustrates the agents along with the capabilities assigned to each one of them with respect to the task Get Info about Care Plan, shown in Figure 3.19.

**Detailed Design**

Detailed design stage aims at specifying agent capabilities and interactions. Thus, during this stage internal and external events that trigger plan and the beliefs involved in agent reasoning are modelled. In our approach we have adapt a subset of the AUML diagrams proposed in [BMO01b]. These are:

- Capability Diagrams. We use AUML activity diagrams to model a capability or a set of capabilities for a specific actor. In each capability diagram, the starting state is represented by external events, activity nodes model plans, transition arcs model events, and beliefs are modelled as objects. An example of a capability diagram is shown in Figure 3.20, in which the allow (or deny) access to services capability of the Authorisation Granter Agent is illustrated. The Authorisation Granter initially
<table>
<thead>
<tr>
<th>Agent</th>
<th>Capabilities</th>
</tr>
</thead>
</table>
| Authorisation Granter      | Get User Authorisation Details  
                            | Get User Authorisation Privileges  
                            | Allow Access to Services  
                            | Deny Access to Services |
| Authorisation Checker      | Provide Info about Users Authorisation Privileges  
                            | Check User Authorisation Status |
| Older Person               | Provide Information about Older Person  
                            | Provide Authorisation Details of Older Person  
                            | Provide Service Description  
                            | Get Older Person query  
                            | Act on Behalf of Older Person |
| Care Planner               | Provide Service Description  
                            | Provide available Care Plan Information  
                            | Re-direct User to Appropriate service Manager |
| Future Actions Agent       | Get Older Person query about Future Actions  
                            | Get Query Results  
                            | Provide Info about Future Actions |
| Professionals Agent        | Get query about Professionals  
                            | Get Query results  
                            | Provide info about Professionals |
| Previous Assessments Agent | Get Query about Previous Assessments  
                            | Get Query results  
                            | Provide Info about Previous Assessments |
| Appointments Agent         | Get Query about Appointments  
                            | Get query results  
                            | Provide Info about Appointments |
| Professional               | Act on Behalf of Professional  
                            | Provide Service Description  
                            | Provide Info about the Professional  
                            | Provide Authorisation Details of the Professional  
                            | Get Professionals query |
| Services Facilitator       | Give access to Medical Libraries  
                            | Give access to Medical Records  
                            | Obtain request for Appointment  
                            | Provide Service Description  
                            | Get request for Service |

Table 3.4: Agent Types and their Capabilities w.r.t. Extended Diagram of Figure 3.19.
accepts an authorisation request from the user. It compares the user’s authorisation details with the user authorisation status that exist in the system, and either allow or deny access to services.

- Plan Diagrams. Plan Diagrams are used to further specify each plan node of a capability diagram. Figure 3.21 illustrates a plan diagram for the Accept Authorisation Request from User plan. The Older Person agent sends an authorisation request to the Authorisation Granter. The Authorisation Granter checks the authorisation request integrity, by interacting with the Integrity Manager, and if the integrity information is valid the Authorisation Granter accepts the authorisation request from the user and acknowledges the authorisation request, otherwise it rejects the authorisation request and notify user about the rejection.

- Agent Interaction Diagrams. We apply in our case sequence diagrams modelling agent Interaction Protocols as proposed by [OB99]. An example of an Agent Interaction Diagram is shown in figure 3.22. The Older Person sends an Authorisation Request to the Authorisation Granter who acknowledges the request. Then the Older Person sends the user’s authorisation details to the Authorisation Granter who forwards them to the Authorisation Checker. The Authorisation Checker checks the user’s Authorisation Status and reply to the Authorisation Granter, who grants or refuses access.
Figure 3.20: Capability Diagram for the Allow or Deny Access capability.

Figure 3.21: Plan Diagram for the Accept Authorisation Request from the User.
Figure 3.22: Agent Interaction Diagram for the Authorisation Procedure.
Chapter 4

Multi-Agent Secure Interactions

4.1 Access control with Security Policies

Using Datalog and logic programs for security policy is customary in computer security [BCFP01, LGF03, BS00, BFA99] and our formal model is based on normal logic programs under the stable model semantics [Apt90]. We have predicates for requests, credentials, assignments of users to roles and of roles to services, see Figure 4.1. They are self explanatory, except for role dominance: a role dominate another if it has more privileges. We have constants for users identifiers, denoted by User:U, for roles, denoted by Role:R, and one for services, denoted by WebServ:S.

Each partner has a security policy for access control $\mathcal{P}_A$ and a security policy for disclosure control $\mathcal{P}_D$. The former is used for making decision about access to the services offered by the partner. The latter is used to decide the credentials whose need can be potentially disclosed to the client.

We keep a set of active (unrevoked) credentials $C_p$ presented by the client in past requests to other services offered by the same server, and the set of declined credentials $C_N$ compiled from the client’s past interactions. To request a service the client submit a set of presented credentials $C_p$, a set of revoked credentials $C_R$ and a service request $r$. We assume that $C_p$ and $C_R$ are disjoint. In this context, $C_N$ is assigned the difference between the missing credentials $C_M$, the client was asked in the previous interaction, and the ones presented now. For stateful autonomic nodes we’ll also need the history of access to services $\mathcal{H}$.

4.2 A Running Example

Let us assume that we have a Planet-Lab shared network between the University of Trento and Fraunhofer institute in Berlin in the context of the E-NEXT network, and that there
are three main access types to the resources: read – access to data residing on the Planet-Lab machines; run – access to data and possibility to run processes on the machines; and configure – including the previous two types of accesses plus the possibility of configuring network services on the machines. All Planet-Lab credentials (certificates) are signed and issued by trusted authorities and the crypto validation is performed before the actual access control process. In other words, a preprocessing step validates and transforms the certificates into a form suitable for the formal model – credential (User : U, Role : R).

Fig. 4.2 shows the role hierarchy, where higher the role in the hierarchy, more powerful it is. A role dominates another role if it is higher in the hierarchy and there is a direct path between them. Fig. 4.3 shows the access and disclosure policies. authNetwork (IP, DomainName) is domain specific: the first argument is the IP address of the authorized network endpoint (the client’s machine) and the second one the domain where the IP address comes from.

Example 1 Rules (1,2) give access to the shared network content to everybody from UniTrento and Fraunhofer, regardless of IP and role. For rules (6,7), if a user has got a disk access and is a researcher at UniTrento or junior researcher at Fraunhofer, it has additional rights. Rules (10,11) give full access from anywhere only to members of the board of directors and to full professors.
Example 2 Rule (5) relaxes the previous two and allows access from any place of the institutions provided users declare their ID and present some role-position certificate of their organization or at least a Planet-Lab membership credential.

Example 3 Rules (1,2) in the disclosure policy show the need for the client to declare its ID if the same comes from an authorized network of the respective organizations; rule (3) discloses the need for Planet-Lab membership credential if the client has already declared its ID; and rule (4) discloses (upgrades) the need of a higher role-position credential.

4.3 Deduction vs Abduction

The basic reasoning service for policy-based approaches is deduction:

Definition 1 (Logical Consequence and Consistency) We use the symbol $P \models L$, where $P$ is a policy and $L$ is either a credential or a service request, to specify that $L$ is a logical consequence of a policy $P$. $P$ is consistent ($P \not\models \bot$) if there is a model for $P$.

This reasoning service is used in most logical formalizations [DS01]: if the request $r$ is a consequence of the policy and the credentials (i.e. $\mathcal{P}_A \cup \mathcal{C}_p \models r$), then access is granted otherwise it is denied.

Example 4 A request coming from doctrorati.dit.unitn.it with IP 193.168.205.11 for access to a fellowship application form on the subnet is granted by rule (3).

The next service is abduction: given a policy and a request, find the credentials that added to the policy would allow to grant the request.
Access Policy:

1. assign (*, request(read)) ← authNetwork (*, *.unitn.it).
2. assign (*, request(read)) ← authNetwork (*, *.fraunhofer.de).
3. assign (*, request(execute)) ← authNetwork (193.168.205, *, *.unitn.it).
4. assign (*, request(execute)) ← authNetwork (198.162.45, *, *.fraunhofer.de).
5. assign (User, request(execute)) ← assign (User, request(read)), declaration (User), credential (User, Role), Role ≥ memberPlanetLab.
6. assign (User, request(addService)) ← assign (User, request(execute)), declaration (User), credential (User, Role), Role ≥ researcher.
7. assign (User, request(addService)) ← assign (User, request(execute)), declaration (User), credential (User, Role), Role ≥ juniorResearcher.
8. assign (User, request(addService)) ← authNetwork (*, *.it), declaration (User), credential (User, Role), Role ≥ assProf.
9. assign (User, request(addService)) ← authNetwork (*, *.de), declaration (User), credential (User, Role), Role ≥ seniorResearcher.
10. assign (User, request(addService)) ← authNetwork (*, *), declaration (User), credential (User, Role), Role ≥ fullProf.
11. assign (User, request(addService)) ← authNetwork (*, *), declaration (User), credential (User, Role), Role ≥ boardOfDirectors.

Release Policy:

1. declaration (User) ← authNetwork (*, *.unitn.it).
2. declaration (User) ← authNetwork (*, *.fraunhofer.de).
3. credential (memberPlanetLab, User) ← declaration (User).
4. credential (RoleX, User) ← credential (RoleY, User), RoleX ⊃ RoleY.

Figure 4.3: Proxy Access and Release Policies for the Online Library

**Definition 2 (Abduction)** The abductive solution over a policy P, a set of predicates (credentials) H (with a partial order ≺ over subsets of H) and a ground literal L is a set of ground atoms E such that: (i) E ⊆ H, (ii) P ∪ E |= L, (iii) P ∪ E ⊬ ⊥, (iv) any set E' ≺ E does not satisfy all conditions above.

Traditional p.o.s are subset containment or set cardinality. Other solutions are possible with orderings over predicates.

This reasoning service is used in the overall interactive access control algorithm shown in Fig. 4.4. Initially the client will send a set of client’s credentials C_p and a service request r. Then we update client’s profile, i.e. declined and active credentials and check whether the active credentials unlock r according to P_A. In the case of denial, we compute all credentials disclosable from C_p according to P_D and from the resulting set remove all C_N. Then we compute all possible subsets of C_D that are consistent with the access policy P_A and, at the same time, grant r. Out of all these sets (if any) the algorithm selects the minimal one. We point out that the minimality criterion could be different for different
Global vars: $C_N, C_P$;  
Internal input: $P_A, P_D$;  
Output: grant/deny/ask($C_M$).

1. client’s input: $C_p$ and $r$,  
2. update $C_N = (C_N \cup C_M) \setminus C_p$, where $C_M$ is from the last interaction,  
3. update $C_P = C_P \cup C_p$,  
4. verify that the request $r$ is a security consequence of the policy access $P_A$ and presented credentials $C_P$, namely $P_A \cup C_P \models r$ and $P_A \cup C_P \not\models \bot$.  
5. if the check succeeds then return grant else  
   (a) compute the set of disclosable credentials $C_D$ as $C_D = \{ c \mid \text{c credential that } P_D \cup C_P \models c \} \setminus (C_N \cup C_P)$,  
   (b) use abduction to find a minimal set of missing credentials $C_M \subseteq C_D$ such that both $P_A \cup C_P \cup C_M \models r$ and $P_A \cup C_P \cup C_M \not\models \bot$,  
   (c) if no set $C_M$ exists then return deny else  
   (d) return ask($C_M$) and iterate.

Figure 4.4: Interactive Access Control Algorithm

Remark 1 Using declined credentials is essential to avoid loops in the process and to guarantee the success of interaction in presence of disjunctive information.

For example suppose we have alternatives in the partner’s policy (e.g., “present either a VISA or a Mastercard or an American Express card”). An arbitrary alternative can be selected by the abduction algorithm and on the next interaction step (if the client has declined the credential) the abduction algorithm is informed that the previous solution was not accepted.

Example 5 Assuming the access and release policies in Figure 4.3, let us play the following scenario. A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions, of a workshop. The service is part of a big management system hosted at the University of Trento’s network that is part of Planet-Lab. So, for doing that, at the time of access, he presents his employee membership token, issued by a Fraunhofer certificate authority, presuming that it is enough as a potential customer.
Formaly speaking, the request comes from a domain \textit{fokus.fraunhofer.de} with credential for \textit{Role: employee} together with a declaration for a user ID, \textit{John Milburk}. According to the access policy the credentials are not enough to get full access and so the request would be denied.

Then, following the algorithm in Figure 4.4, it is computed the set of disclosable credentials from the disclosure policy and the user’s available credentials, and the minimal set of credentials, out of those, that satisfies the request. The resulting set is \{credential (User: \textit{John Milburk}, Role: \textit{junior Researcher})\}. Then the need for this credential is return back to the user.

\textbf{Example 6} \textit{On the next interaction step, because the user is a senior researcher, the same declines to present the requested credential as just returning the same query with no presented credentials.}

So, the algorithm updates the user’s session profile and the outcome is the need for credential \{credential (User: \textit{John Milburk}, Role: \textit{senior Researcher})\}.

\section*{4.4 Stateful AC: Missing and Exceeding Credentials}

What happens if access to services is determined also by the history of past executions? For instance in the example by Atluri and Bertino \cite[pag.67]{BFA99} a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque. So, if we have no memory of past credentials then it is impossible to enforce any security policy for separation of duties on the application workflow. The problems are the following:

- the request may be inconsistent with some role used by the client in the past;
- the new set of credential may be inconsistent with requirements such as separation of duties;
- in contrast to intra-enterprise workflow systems \cite{BFA99}, the partner offering the service has no way to assign to the client the right set of credentials which would be consisted with his future requests (because he cannot assign him future tasks).

So, we must have some roll-back procedure by which, if the user has by chance sent the “wrong” credentials, he can revoke them.

Our interactive access control solution for stateful services and applications is shown in Figure 4.5.

The logical explanation of the algorithm is the following. Initially when a client requests a specific service the authorization mechanism creates a new session with global variables declined credentials $C_N$, not revoked credentials $C_U$, missing credentials $C_M$ and
Global vars: $C_N, C_U, C_M, C_E$; Initially $C_N = C_U = C_M = C_E = \emptyset$;

Internal input: $\mathcal{P}_A, \mathcal{P}_D, \mathcal{H}, C_P$;

Output: grant/deny/$<\text{ask}(C_M), \text{revoke}(C_E)>$;

1. **client’s input:** $C_p, C_r$ and $r$,

2. update $C_P = (C_P \setminus C_r) \cup C_p$,

3. update $C_N = (C_N \cup C_M) \setminus C_p$, where $C_M$ is from the last interaction,

4. update $C_U = (C_U \cup C_E) \setminus C_r$, where $C_E$ is from the last interaction,

5. Set up $C_M = C_E = \emptyset$,

6. verify whether the request $r$ is a security consequence of the policy access $\mathcal{P}_A$ and presented credentials $C_P$, namely $\mathcal{P}_A \cup \mathcal{H} \cup C_P \models r$ and $\mathcal{P}_A \cup \mathcal{H} \cup C_P \not\models \bot$,

7. if the check succeeds then return **grant** else

   (a) compute the set of **disclosable credentials** $C_D = \{c \mid \mathcal{P}_D \cup C_P \models c\} \setminus (C_N \cup C_P)$,

   (b) use abduction to find a minimal set of **missing credentials** $C_M \subseteq C_D$ such that both $\mathcal{P}_A \cup \mathcal{H} \cup C_P \cup C_M \models r$ and $\mathcal{P}_A \cup \mathcal{H} \cup C_P \cup C_M \not\models \bot$,

   (c) if a set $C_M$ exists then return $<\text{ask}(C_M), \text{revoke}(C_E)>$ else

      i. use abduction to find a minimal set of missing credentials $C_M \subseteq (C_D \cup C_P)$ such that $\mathcal{P}_A \cup \mathcal{H} \cup C_M \models r$, $\mathcal{P}_A \cup \mathcal{H} \cup C_P \not\models \bot$ and $C_U \cap (C_P \setminus C_M) = \emptyset$,

      ii. if no set $C_M$ exists then return **deny** else

      iii. compute $C_E = C_P \setminus C_M$ and $C_M = C_M \setminus C_P$,

      iv. return $<\text{ask}(C_M), \text{revoke}(C_E)>$ and iterate.

---

Figure 4.5: Interactive Access Control Algorithm for Stateful Autonomic Services

excessing credentials $C_E$ set up to empty sets. Then once the session is started, internally, the algorithm loads the policies for access and disclosure control $\mathcal{P}_A$ and $\mathcal{P}_D$ together with the two external sets history of execution $\mathcal{H}$ and client’s active credentials $C_P$.

Following that, the first step in the algorithm is to get the client’s input as sets of currently presented credentials $C_p$, the revoked ones $C_r$ and the service request $r$. Then the set of active credentials $C_P$ is updated as removing the set $C_r$ from it and then adding the set of currently presented credentials (rf. step 2). Then in step 3 declined credentials $C_N$ are updated as credentials the client was asked in the last interactions minus the ones that he has currently presented. Analogously, in step 4, not revoked credentials $C_U$ are updated as the excessing credentials asked in the last interaction minus the ones currently revoked. Step 5 prepares the two sets $C_M$ and $C_E$ for the interaction output.
Steps 6, 7, 7a, 7b and 7c have the same explanation as the respective ones in Figure 4.4. If a set of missing credentials was not found in step 7b then we run the abduction process again (step 7(c)i) but over the extended set of disclosable credentials and active credentials \( \mathcal{C}_D \cup \mathcal{C}_D \) searching for a solution for \( r \) that preserves consistency in \( \mathcal{P}_A \) and unlocks \( r \). The last requirement in the step is used to filter out those solutions that have been partially refused to be revoked.

Step 7(c)i indicates that if a set \( \mathcal{C}_M \) exists then definitely there are “wrong” credentials among those in \( \mathcal{C}_P \) that ban the client to get a solution for \( r \) (in step 7b). If no such set then the client is denied because he does not have enough privileges to disclose more credentials to obtain the service \( r \) (step 7(c)ii).

Step 7(c)iii computes the sets of excessing and missing credentials \( \mathcal{C}_E \) and \( \mathcal{C}_M \). The motivation behind \( \mathcal{C}_E \) is that the set difference of active credentials minus just computed \( \mathcal{C}_M \) certainly contains the credentials that ban the client to get a solution for \( r \).

At this point there two main issues concerning the set \( \mathcal{C}_E \): (i) the system may restart from scratch asking the client to revoke all his active credentials, i.e. \( \mathcal{C}_E = \mathcal{C}_P \), (ii) the system may ask the client to present credentials that have been already asked for revocation in past interactions.

**Remark 2** Step 7(c)iii looks the opposite of abduction: rather than adding new information to derive more things (the request), we drop information to derive less things (the inconsistency). One can show that the two tasks are equivalent.

### 4.5 Life is complicated: Two-party Negotiation

So far deduction helps us to infer whether a service request is granted by the partner’s access policy and the client’s set of credentials. In the case of failure, abduction infers what is missing so that the client can still get the desired service.

**Example 7** *When the senior researcher received the counter request to present his seniorResearcher certificate in order to get access he may not want to reveal his role if he is not sure that he talks with a University of Trento’s server.*

We should allow him to request the system to show a certificate. The system, in its turn, may have policy saying that such certificates are disclosed only to entities coming from an authorized network, e.g., \( \text{authNetwork} (\ast, \ast, \text{fraunhofer.de}) \).

The next step is how to establish and automate a two-party negotiation process using the inference capabilities on both sides. For that purpose we need to extend each of the party’s policies:

- a policy for access to own resources \( \mathcal{P}_{AR} \) on the basis of foreign credentials,
a policy for access to *own* credentials $\mathcal{P}_{AC}$ on the basis of *foreign* credentials,

- a policy for the disclosure of the need of missing *foreign credentials* $\mathcal{P}_D$.

Client and the server just have to run the same negotiation protocol:

1. The client, Alice, sends a service request $r$ and (optionally) a set of credentials $C_p$ to the server, Bob.

2. Then Bob looks at $r$ and if it is a request for a service he calls the interactive access control algorithm in Figure 4.4 with his policies for access and disclosure of resources $\langle \mathcal{P}_{AR}, \mathcal{P}_D \rangle$.

3. If $r$ is a request for a credential then he calls the same algorithm with his respective policies for access and disclosure of credentials $\langle \mathcal{P}_{AC}, \mathcal{P}_D \rangle$.

4. In the case of computed missing credentials $C_M$, he transforms that into counter-requests for credentials and waits until receives all responses. At this point Bob acts as a client, requesting Alice the set of credentials $C_M$. Alice will run the same protocol swapping roles.

5. When Bob’s main process receives all responses it checks whether the missing credentials have been supplied by Alice.

6. If $C_M$ was not reached, Bob restarts the loop and consults the interactive access control algorithm for a new decision.

7. When a final decision is taken, the response (grant/deny) is sent to Alice.

The protocol can be run on both sides so that they can communicate and negotiate the missing credentials until enough trust is established and the service is granted or the negotiation failed and the process is terminated.

### 4.6 Induction: Finding the rules

The work for inductive logic programming [MDR94] has been most evolved in the field of machine learning. Inductive logic programming systems (ILP) construct concept definitions from examples and a logical domain theory.

Induction may be an extremely valuable tool for autonomic nodes, because complete and consistent access policies may be difficult to write. So it might be well the case that a node has only a partial policy, and some additional set of examples of access that one desired to permit or forbid. Then the node should be able, by generalizing from the examples to derive a policy that matched the given examples and is also able to answer other similar queries.
So an autonomic node could be provided with background police \( P_B \), some sample granted requests for services \( R^+ \) and denied requests \( R^- \) and as a result it should be able to constructs a tentative access policy \( P_{HA} \). Here \( R^+ \), \( R^- \) are sets of ground facts and \( P_B \) and \( P_{HA} \) are logic programs. The conditions for construction of \( P_{HA} \) are:

**Necessity:** \( P_B \not\models R^+ \),

**Sufficiency:** \( P_B \land P_{HA} \models E^+ \),

**Weak consistency:** \( P_B \land P_{HA} \not\models \bot \),

**Strong consistency:** \( P_B \land P_{HA} \land E^- \not\models \bot \).

A number of algorithms can be used for determining the construction of \( P_{HA} \) based on ILP (see e.g. [MDR94]) and the identification of the most appropriate for autonomic communication policies is the subject of future work.

### 4.7 Implementation

We have implemented a system for access control for abduction and deduction using protocols over web services, a front-end to a state-of-the-art inference engine and integrated it with a system for PMI (privilege management infrastructure).

For our implementation, Collaxa\(^1\) is used as a main manager of Web Services Business Processes (on the AuthorizationServer side).

**PolicyEvaluator** is a Java module that acts as a wrapper for the DLV system\(^2\) (a disjunctive datalog system with negations and constraints) and implements our interactive algorithm for stateless autonomic nodes (Fig. 4.4). For deductive computations we use the disjunctive datalog front-end (the default one) while for abductive computations, the diagnosis front-end.

The current system processes credentials at an high level: defines what can be inferred and what is missing from a partner’s access policy and a user’s set of credentials. For the actual distributed management of credentials at lower levels (namely actual cryptographic verification of credentials) we decided to use PERMIS infrastructure [CO02] because it incorporates and deals entirely with X.509 Identity and Attribute Certificates. It allows for creating, allocating, storing and validating such certificates. Since PERMIS conforms to well-defined standards we can easily interoperate with the other entities (partners) in the network.

---

\(^1\) Collaxa BPEL Server (v2.0 rc3) – www.collaxa.com

\(^2\) DLV System (r. 2003-05-16) – www.dlvsystem.com
4.8 The Challenges Ahead

So far we have presented a logical framework and a proof-of-concept implementation for reasoning about access control for autonomic communication based on interaction for supplying missing credentials or for revoking “wrong” credentials. We have discussed the different formal reasoning services – deduction, abduction, and induction, with a special emphasis of the first two. We have also show how the model can deal with stateful access to services and two party negotiation.

Yet, a number of major challenges remains:

**Complexity Characterization:** abduction engines such as DLV are rather effective but unfortunately general algorithms for abduction are inefficient\(^3\). Our problems are at the same time more specialized (e.g. credentials are occurring only positively in the rules) and more general (we have hierarchies of roles so subset or cardinality minimality does not really apply). So capturing the exact computational complexity of the problem may be far from trivial.

**Approximation vs Language Restriction:** even if the problem is hard in the general case, we might have suitable syntactic restrictions that allow for a polynomial evaluation. In other cases, we may be able to find out anytime algorithms that gives an approximate answer (not really the minimal one but close to it).

**Reputation Management:** so far we have assumed that declining or presenting a credential has no impact on the reputation of nodes. Research on algorithms and logics for secure reputation is still in the early stage but its integration with interactive access control might have a significant impact.

**Negotiation Strategy Analysis:** which is the impact of the negotiation strategy on the effectiveness, completeness, privacy protection, immunity from DoS attacks of interactive access control? So far only the completeness of the procedure is settled and more sophisticated strategies, taking into account the value of credentials that are disclosed could lead to many interesting results relevant for the practical deployment of the framework.

**Policy Compilation:** this is likely the topic with major impact on industry. All policies (either in networking or security) are either interpreted or hard-wired in the application. In contrast, we would need a way to “compile” the policy and the policy enforcement engine into machine languages so that autonomic nodes can quickly react to the requests and yet gives us the flexibility of policies: update a policy simply means recompiling and redeploying.

---

\(^3\)They lay at the secon level of the polynomial hierarchy, i.e. harder than NP.
Chapter 5

Multi-Agent Databases

5.1 Towards Knowledge-Driven Peer-to-Peer computing

In the Peer-to-Peer (P2P) model computational peers interoperate in a completely decentralized distributed environment, providing to and requesting from each other data and/or services. Peers are largely autonomous in what and how they store in their local (knowledge) bases, in what data and services they provide to other peers, in what other peers they “talk to”, etc. Peers come and go, make spontaneous acquaintances with other peers, and, eventually, drop them. Peers collaboratively process user requests, and the overall performance of the network emerges from local point-to-point interactions of (all) peers on the network.

P2P applications cover a number of domains such as file sharing (e.g., Kazaa\(^1\)), distributed computing (e.g., SETI@Home\(^2\)), collaborative networking (e.g., Groove\(^3\)), and instant messaging (e.g., ICQ\(^4\)). However, most of the P2P systems are hybrid - peers, (mainly) for discovery purposes, rely on a centralized resources indexing server. An example of a totally decentralized P2P application is Gnutella\(^5\), which is a file sharing application. In file sharing applications, all peers rely on the same schema that describes the content of the files they share. What here needs to be underlined is that, although technologically distributed, P2P (data sharing) systems are at least conceptually centralized as far as they have to assume some shared semantics in order to exchange data meaningfully (e.g. names and meaning of categories). For a comprehensive overview of the P2P technology and applications see, for instance, [SVR\(^+\)02].

According to some studies carried out at Gartner, P2P has passed its ”peak of inflated expectations”, and now true technology’s applicability, risks and benefits need to be un-

\(^1\)Kazaa: http://www.kazaa.com
\(^2\)SETI@Home: http://setiathome.ssl.berkeley.edu
\(^3\)Groove: http://www.groove.net
\(^4\)ICQ: http://www.icq.com
\(^5\)Gnutella: http://www.gnutella.com
At this stage, exploration of application domains where P2P can better exploit its technological potential becomes vital. According to the Gartner report, it will take from 5 to 10 years before the real-world benefits of the P2P technology are demonstrated and accepted.

On the other hand, the emerging Semantic Web (SW) technologies open new horizons for P2P. The vision of the SW is to enable machines to retrieve and process meta-data (i.e., information about data), and exploit this knowledge for intelligent, meaningful and context-driven interoperation with users and other applications. In the SW P2P scenario, users encode their knowledge in a formal structure, such as ontology, and then share it with other users and applications; they create communities of knowing which gradually evolve as new knowledge is brought in. Moreover, the SW allows it to overcome current limitations of P2P data sharing applications. Namely, it allows for richer and mutually heterogeneous peers’ schemas, while still ensuring interoperability. Note that when peers’ schemas are heterogeneous, the role of centralized resource indexing servers diminishes, as there is no common schema to index resources.

Another knowledge-intensive domain is (organizational) Knowledge Management (KM). There is very little in the literature about what P2P can do for KM. However, Ovum reports that “Of all the application domains we have studied, knowledge management is the one where the benefits of peer-to-peer and a clear and straightforward business model for suppliers are most evident” [CRNE02]. Ovum identifies at least two major areas where P2P can have impact on KM: collaboration and knowledge discovery. The former area includes the support for virtual teams and organizations, and for communities of practice. The latter one is about finding content, social patterns and specialists’ profiles in enterprise and personal P2P networks. At the technical level, these tasks require expressive formalisms to represent and share knowledge. From this point of view, SW technologies become very useful.

Currently, none of the commercialized P2P applications can be seen as a comprehensive KM solution. However lots of research is being done on this topic in academia. For instance, the Semantic Web community discusses the role of ontologies in KM systems (e.g., [MCJ03, DSRvHF02]). The database community proposes several solutions for a completely decentralized P2P database system with the support of heterogeneous schemas [PFA02, YIPI03]. There are many other areas that contribute to building viable P2P KM solutions - personal knowledge management [E02], semantic matching of ontological structures to facilitate peers interoperability (see [FP03]), and so on.

---

6Gartner Hype Cycle 2002 - Information Technology Trends
7For more information on the Semantic Web, see the W3C Semantic Web activity page: http://www.w3.org/2001/sw/
5.2 Towards Distributed Knowledge Management

The traditional architecture of KM systems have embodied the assumption that, to share and exploit knowledge, it is necessary to implement a process of knowledge-extraction-and-refinement, whose aim is to eliminate all subjective and contextual aspects of knowledge, and create an objective and general representation that can then be reused by other people in a variety of situations [MPA00, MPP02, MPR02]. Very often, this process is finalized to build a central knowledge base, where knowledge can be accessed via a knowledge portal. This centralized approach - and its underlying objectivist epistemology - is one of the reasons why so many KM systems are deserted by users, who perceive such systems either as irrelevant or oppressive (see [MD01, HE01]). As clearly pointed by the Report on Knowledge Management to the European Commission, 2004 "KM is a crucial competence in the new competitive arena but the degree of predictability which has been inherent in KM thinking, reflecting the general belief in linearity, is now seriously questioned."

During the last year, the evidence that knowledge is a distributed, contextual and subjective matter have led to an alternative vision, the so called Distributed Knowledge Management (DKM). As described in [MPP02], DKM is an approach to KM based on the principle that the multiplicity (and heterogeneity) of perspectives within complex organizations should not be viewed as an obstacle to knowledge exploitation, but rather as an opportunity that can foster innovation and creativity.

The fact that different individuals and communities may have very different perspectives, and that these perspectives affect their representation of the world (and therefore of their work) is widely discussed - and generally accepted - in theoretical research on the nature of knowledge. Knowledge representation in artificial intelligence and cognitive science has produced many theoretical and experimental evidences of the fact that what people know is not a mere collection of facts; indeed, knowledge always presupposes some (typically implicit) interpretation schema, which provides an essential component in sense-making (see, for example, the notions of context [J.93, P.98, CF01], mental space [G.85], partitioned representation [J.91]). Moreover, studies on the social nature of knowledge stress the social nature of interpretation schemas, viewed as the outcome of a special kind of "agreement" within a community of knowing (see, for example, the notions of scientific paradigm [T.79], frame [K.95], thought world [D.92], perspective [JR95]). In this sense, rather than linear, knowledge dynamics are better represented by evolving constellations of autonomous and heterogeneous "knowledges" that consolidate their knowledge locally and seek for some form of coordination with other "knowledges" by means of semantic negotiation.

Despite this large convergence, the need to preserve an idea of control which is inherent to the very notion of business resource, led organizations and managers to neglect what increasingly happens behind the scene of any corporate intranet: people and groups still continue to develop an interrelated web of local systems that better fit their needs. On
the other hand, the risk that these constellations become unmanageable, and the maturity that is being reached by P2P technologies are increasingly attracting managers towards alternative perspectives. Among these, at least the opportunity to weakly control knowledge constellations making them visible, instead of hiding them behind the official but ephemeral claims of control provided by the knowledge portal. In this less ideal but more realistic landscape, P2P systems seem particularly suitable to implement this view in which the existence of knowledge ”lobbies” is recognized and inter-community cooperation is supported through the provision of adequate coordination facilities. In the next section, we propose a system that tries exactly to go in this direction.

5.3 Peer-to-Peer Knowledge Management - Prospects and an Application

The convergence of P2P and KM technologies creates new challenges for researchers to address: new methodologies to model, design, and deploy distributed KM solutions; theories and algorithms to represent the social and semantic dimensions of a knowledge network; mechanisms to cope with the dynamic autonomous nature of P2P and to provide means to support emergent network self-organization. New technologies should be provided in order to support full operational functioning of P2P KM systems, ensuring high extensibility of the solutions along several dimensions, such as scalability in the number of peers, size and kind of supported knowledge bases, level of heterogeneity in knowledge representation, etc.

In return, P2P KM applications give the prospects of robustness, large pool of shared resources and semantics-driven tools to effectively operate these resources; local autonomy and comprehensive support to each peer provided by a collaborative effort of other peers. P2P KM solutions are not dependent on the presence of certain peers or content on the network; instead, peers bring new knowledge which flows along semantic links between peers, being enriched and completed on the fly. The ”knowledge base” of a P2P KM network is formed dynamically; peers forge and break knowledge groups based on a common interest, etc. From this point of view, it turns out that operation on a P2P KM network naturally complements usual economic and social patterns.

As an example of an existing P2P KM system, we propose in this chapter a P2P DKM architecture, named KEEx, which is the result of the research project EDAMOK⁸ (Enabling Distributed and Autonomous Management of Knowledge). In KEEx, each community of knowing (or Knowledge Nodes (KN), as they are called in [MPR02]) is represented by a peer, and the two principles above are implemented in a quite straightforward way: (i) each peer provides all the services needed by a knowledge node to create and organize its own local knowledge (autonomy), and (ii) by defining social structures and protocols of meaning negotiation in order to achieve semantic coordination (e.g., when

⁸EDAMOK: http://edamok.itc.it
searching documents from other peers). Built on top of this architecture, distributed local "knowledges" can emerge and aggregate through a bottom-up process from the individual level, to the organizational one, passing through the establishment of communities (group of peers that share a similar interest) and zones (networks of peers that relate to a neighbourhood). Moreover, peers' knowledge bases can be run on top of industrial solutions such as existing content management applications or databases. As a consequence, knowledge bases become to be virtual, flexible and temporary aggregation of both individual and more institutionalized knowledge sources.

Figure 5.1: The KEEx's main components.

The main components of KEEx are shown on Figure 5.1. Each Knowledge Node (also known as K-peers) can play two main roles: provider and seeker. In the former case, a K-peer "publishes" in the system a body of knowledge, together with an explicit representation on it. In the latter case, a K-peer searches for information by explicitly specifying a query as a part of its own perspective. K-peers store their local knowledge in document repositories. K-peers formally describe the real world (from their own perspective) in the approximate and partial form of a context. Contexts are also used by seekers for query representation. A K-peer may have more than one context, and it stores its context(s) in a context repository. Contexts are created, manipulated and used by K-peers by means of the context management module, which includes a context editor and a context browser. Apart from this, KEEx allows for semantic matching of contexts, for forming federations of K-peers based on knowledge that the peers have in common, and for peers discovery.
KEEx is implemented on top of the P2P platform JXTA⁹. For a throughout discussion of KEEx’s components and functionality, see [MPGM02].

⁹JXTA: http://www.jxta.org
Chapter 6

Conclusion

In this deliverable we have presented results from our work to extend the Tropos methodology to enable it to consider security requirements throughout its development stages. During the process of extending Tropos some very useful observations were obtained. Firstly, the concept of constraints is a natural extension of the Tropos methodology and it allows for a systematic approach towards the modelling of security requirements. This is because, although functional and security requirements are defined alongside, a clear distinction is provided. Secondly, the security diagram allows identifying desired security requirements very early in the development stages, and helps to propagate them until the implementation stage, introducing a security-oriented paradigm to the software process. In addition, the iterative nature of the methodology, allows the re-definition of security requirements in different levels therefore providing a better integration with system functionality.

We proposed an approach for the development of secure information systems that merges two important software engineering paradigms: agent-oriented software engineering (AOSE) and patterns. We believe that the integration of these two paradigms provides a complete and mature solution for the development of secure information systems. We consider the integration to be complete, since we believe that those paradigms complement each other:

- AOSE provides concepts and notations suitable for modeling security issues in information systems, such as autonomy, intentionality and sociality.

- Patterns complement agent-oriented techniques by transferring security knowledge to non-security application experts in an efficient manner.

We also consider the integration to be mature as such an integration will make security solutions more widely available, providing novice and non-security expert developers with the capability of implementing secure information systems.
We proposed two applications of security concepts to real case studies. In particular, we applied the Tropos methodology in the analysis and design of the electronic Single Assessment Process (eSAP), an electronic system to deliver the integrated health assessment of health and social care needs of older people. The main conclusions derived from this attempt were encouraging. Also, we showed the Secure Tropos methodology at work on a real-life comprehensive case study encompassing an ISO-17799 security management policy. The proposed constructs and methodology were up the challenge and revealed a number of pitfalls, especially when the formal analysis techniques were applied.

We proposed a logical framework and a Web-Service based implementation for reasoning about access control for Autonomic Communication. Our model is based on interaction and exchange of requests for supplying or declining missing credentials. We identify the formal reasoning services that characterize the problem and sketch their implementation.

Finally, we argued that technological architectures, when dealing with processes in which human communication is strongly involved, must be consistent with the social architecture of the process itself. In particular, in the domain of KM, technology must embody a principle of distribution that is intrinsic to the nature of organizational cognition. Here, we suggest that P2P infrastructures are especially suitable for KM applications, as they naturally implement meaning distribution and autonomy. It is worth noting at this point that other research areas are moving toward P2P architectures. In particular, we can mention the work on P2P approaches to the semantic web [MAI02], databases [FI02], web services [MJB03]. We believe this is a general trend, and that in the near future P2P infrastructures will become more and more attractive to all areas where we can’t assume a centralized control.
Chapter 7

History of the Deliverable

Below we outline how the work described in the deliverable has evolved along the years of the project.

Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Author (Unit)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2003/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>First version of using Tropos methodology to model an integrated health assessment system.</td>
</tr>
<tr>
<td>0.2</td>
<td>2003/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>Revision of security patterns of multi-agent systems.</td>
</tr>
<tr>
<td>0.3</td>
<td>2003/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>Revision of modeling secure multi-agent systems.</td>
</tr>
<tr>
<td>0.4</td>
<td>2004/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>Revision of security reference diagrams and security attack scenarios.</td>
</tr>
<tr>
<td>0.5</td>
<td>2004/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>First version of reasoning services for access control in automatic communication.</td>
</tr>
<tr>
<td>0.6</td>
<td>2005/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>Revision of using a security requirements in practice.</td>
</tr>
<tr>
<td>0.7</td>
<td>2005/Nov</td>
<td>Draft</td>
<td>UNITN, ITC</td>
<td>First version of peer-to-peer knowledge management.</td>
</tr>
<tr>
<td>0.8</td>
<td>2006/Nov</td>
<td>Draft</td>
<td>UNITN</td>
<td>Revision of modeling secure systems using an agent-oriented approach.</td>
</tr>
<tr>
<td>1.0</td>
<td>2006/Nov</td>
<td>Final</td>
<td>UNITN</td>
<td>Final reversion of the deliverable.</td>
</tr>
</tbody>
</table>
Bibliography


123


[CLMR02] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, and Joseph Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation, April 2002.


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title</th>
<th>Publication Details</th>
</tr>
</thead>
</table>


