# A framework for defining and analysing access policies in requirements models

## Thesis

# A Framework for Defining and Analysing Access Policies in Requirements Models

**Robert P. Crook** BSc, MBA

Submitted for the degree of Doctor of Philosophy

The Open University

Department of Computing

Faculty of Mathematics and Computing

September 2007



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

# Abstract

Enforcing access policies derived from management control principles is a way by which organisations protect their information assets. The minimum privileges principle is an example of a management control principle, which specifies that users should only have access to resources they require to carry out their duties. Requirements models use actors to specify their access policies. Actors normally represent roles that users adopt, however a role can have different meanings, such as a position in an organisation or the assignment of a task, and can therefore be misleading. Current requirements modelling approaches do not provide a systematic way of defining roles for incorporation into access policies, and therefore we can not ensure that they satisfy management control principles. In this thesis we address the need to provide precise role definitions by developing a framework that facilitates the derivation of roles from the organisational context. The framework consists of a meta-model, which enables the organisational context to be represented and related to actors; a set of heuristics for deriving the organisational context; and a set of language constructs for formulating access policies, and verifying them using scenarios.

We use the meta-model and language constructs that we developed to extend an existing requirements modelling language, the i* framework, and in particular a formal version of it, formal Tropos, to define and verify access policies definitions satisfying the minimum privileges principle. We also investigate the use of automated tool checking by translating the formal Tropos definitions into the specification language Alloy, which is supported by a tool that automatically checks assertions, to

ensure consistency of the access policy definitions. We carry out a detailed case study taken from the literature to verify the extensions to the i* framework and the tool supported analysis.

The framework presented in this thesis makes a novel contribution to the modelling of access policies as requirements, enabling us to define access policies using actors derived from the organisational context, that satisfy the minimum privileges principle.

# Acknowledgements

I would like to thank my supervisors, Bashar Nuseibeh and Darrel Ince, for their tremendous support and advice. I thank members of the Computing Department at the Open University, in particular Jonathan Moffett, who played an important consultative role and reviewed several papers, also Marian Petre, Jon Hall and David Bowers who have reviewed earlier versions of the thesis, and Charles Haley and Luncheng Lin for interesting discussions. I'm also grateful to the examiners, Ali Abdallah and Robin Laney, for their valuable advice on how to improve the technical presentation of the thesis. Finally I would like to thank my father John Crook, my mother Hilary Crook, and my sister Barbara Claessens for their insights into the organisational structures and procedures in hospitals that were helpful in formulating a case study.

# Statement of Contribution

Much of the material in this thesis appears in the following papers and books.

- Crook, R., Nuseibeh, B., Lin, L., and Ince, D. "Security Requirements Engineering: When Anti-Requirements Hit the Fan," IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 11-13th September, 2002, pp. 203-205.
- Crook, R., Ince, D., and Nuseibeh, B. "Towards an Analytical Role Modelling Framework," 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02), Essen, Germany, 9-10th September, 2002, pp. 123-136.
- Crook, R., Ince, D., and Nuseibeh, B. "Modelling Access Policies using Roles in Requirements Engineering," *Information and Software Technology* (45:14), November 2003, pp. 971-991.
- Crook, R., Ince, D., and Nuseibeh, B. "On Modelling Access Policies: Relating Roles to the Organisational Context," IEEE International Requirements Engineering Conference (RE'05), Paris, France, 29th August -1st September, 2005, pp. 157-166.
- Crook, R., Ince, D., and Nuseibeh, B. "Using i* to Model Access Policies: Relating Roles to their Organisational Context," (to appear in) *Social Modelling for Requirements Engineering*, Giorgini, P., Maiden, N., Mylopoulos, J., and Yu, E. (ed.), MIT Press.

The thesis should be regarded as the definitive account of this work. All the work in this thesis describes the original contributions of the author.

# Table of Contents

# Figures and Tables

# Chapter 1

# Introduction

Security incidents can be very costly for organisations; Nick Leeson's unauthorised trading resulted in losses of over £800 million, so causing the bankruptcy of Barings Bank (Brown & Steenbeek, 2001); there are similarities with the case of John Rusnak, who defrauded the Allied Irish Bank of a similar amount in 2002 (Massaci & Zannone, 2006). In both cases the culprits exploited weaknesses in the computer systems designed to control their trading activities. These are prominent examples of a problem highlighted by Anderson (2001) that computer fraud is often caused by staff; i.e. authorised users, accidentally discovering features of a system, and exploiting them. There is a need to keep outsiders from breaking in, but, it is also equally important to prevent users with legitimate authorisation abusing their privileges in the way that Leeson and Rusnak did. Organisations have access policies based on the principles of management control to prevent these sorts of incidents happening. Access policies are the rules, which regulate how users can access resources (Moffett & Sloman, 1988). A problem was that the computer systems that Rusnak and Leeson used, did not adequately enforce these policies.

The focus of our research is on the modelling of security requirements based on the principles of management control. Nuseibeh & Easterbook (2000) provide an apt description of what we mean by modelling as "the construction of abstract descriptions that are amenable to interpretation". In requirements models, users are

represented by actors, which usually describe a role they are undertaking such as carrying out a task, or a position in an organisation. A weakness of current requirements modelling approaches is that they do not allow us to model all aspects of the organisational context and relate this to actors, which is a prerequisite for formulating requirements to enforce policies based on the principles of management control. In this thesis we differentiate between the micro- and macro-levels of the organisation. This follows the convention in the organisational behaviour literature when performing analysis, as exemplified by Rollinson (2005). The micro-level of the organisation is concerned with individuals, groups, and interpersonal relationships, whereas the macro-level of the organisation is concerned with the organisational structure, organisational design, and culture. When referring to the organisational context we therefore differentiate between the micro-organisational and the macro-organisational contexts to reflect these different levels of analysis.

The research we present in this thesis is aimed at strengthening the link between actors and the organisational context, to improve the process of defining access policies in requirements models, through the development of a framework. The framework consists of a meta-model, which enables the organisational context to be represented and related to actors; a set of heuristics for deriving the organisational context; and a set of language constructs for formulating access policies, and verifying them using scenarios.

This chapter outlines the background and motivation behind the work, and defines the objectives of the research. A road map of the thesis is also sketched.

## 1.1 Background and Motivation

The International Organisation for Standardisation code of practice for information security management (ISO, 2005) states that "information is an asset that, like other important business assets is essential to an organisation's business and consequently needs to be suitably protected". Security goals, from which security requirements are derived, are concerned with maintaining the confidentiality, integrity and availability of assets, against the potential harmful actions of users (van Lamsweerde *et al.*, 2003). Goals and requirements can be derived from a threats analysis, where the harmful intent of actors and their actions can be identified, and suitable countermeasures defined. However an additional source of security goals is the set of management control principles (Moffett *et al.*, 2004).

Management control principles are practices applicable to many large organisations, to ensure that employees perform their duties commensurate with the objectives of the organisation, and do not commit fraud. Fraud in commercial organisations is frequently caused by users who abuse their legitimate privileges (Anderson, 2001), a problem that existed before IT was introduced into organisations. Management control principles, originating from legislation, accounting and management practices, are implemented to prevent these sorts of incidents from taking place. These organisational control principles need to be enforced by computer systems used to manage valuable assets, and are translated into access policies (Moffett & Sloman, 1988).

Security engineering researchers, in developing access control solutions, have long since faced the problem of how to define access policies, and as a result there exists an extensive body of literature on this subject. Examples of this include the

definition of security clearances for military or governmental applications (Bell & LaPadula, 1973), separation of duties in commercial applications (Clark & Wilson, 1987), delegation of duties (Moffett & Lupu, 1999; Barka & Sandhu, 2000), and contextually based restrictions (Georgiadis *et al.*, 2001). An access policy can be based on a number of factors, such as membership of a group, the level of authority of an individual, a delegated task, whether this individual can perform other related tasks, temporal and other environmental constraints. Researchers in the security field have found roles a useful way of capturing these factors, and hence to define policies based on them. Access control that uses roles to define policies is Role-Based access control (RBAC) (Sandhu *et al.*, 1996), a role being essentially a collection of permissions, but which map onto organisational roles.

There is an important parallel between, defining actors in a requirements model, and the research into RBAC, in that both are concerned with defining roles. Requirements models represent users as actors or agents that are assigned to actions. This assignment can be used to represent access policies (Liu *et al.*, 2003). An actor represents a role. However a problem arises in that the use of the notion of a role can vary, from the assignment of a task, as proposed by Yu (1997), to a position within an organisational hierarchy (Sandhu *et al.*, 1996). The fact that there is no clear definition as to what a role means can lead to ambiguity; this problem is exemplified by He *et al.* (2006), who found different terms were used to describe the same role.

Within the security research community there are key differences between the way in which researchers propose how roles should be defined to represent the organisation. For example Moffett & Lupu (1999) and Sandhu *et al.* (1996) differ in their views as to how the organisational hierarchy can be modelled using roles. It is not surprising that this is the case when we consider the view from the sociology

literature that in the most general sense a role is a term that describes behaviour (Biddle, 1979) as diverse as an angry parent to a government minister. There are different types of roles, for example there are positional roles, functional roles, and contextual roles (Biddle, 1979); to complicate this further, individuals adopt multiple roles at the same time (Handy, 1985).

Defining a role is therefore difficult (He & Antón, 2003). A role however is a means to an end, and in deriving roles they need to be defined in a way that enable us to derive access policies that satisfy the principles of management control (Moffett & Lupu, 1999), which entails relating them to the organisational context.

Most requirements modelling approaches, such as GBRAMS (Antón, 1996), KAOS (Dardenne *et al.*, 1993), Use Cases (Cockburn, 2001), and CREWS (Maiden, 1998) do not relate actors to the organisational context in that they do not show to which part of the organisation they are assigned, and what level of seniority they have.

However, there is a class of requirements modelling approaches that do derive requirements from the organisational context: the i* framework (Yu, 1997), ORDIT (Dobson *et al.*, 1992), and an enterprise modelling approach proposed by Loucopoulus & Kavakli (1995). Nevertheless these modelling approaches have significant weaknesses with regard to their use in defining access policies. The i* framework, a goal-based modelling framework for representing dependencies between actors, tasks and resources, focuses mainly on individuals and their intentions in a social setting; i.e. the micro-organisational context. However access policies include the macro-organisational context; i.e. the way groups are structured, and the power structures that determine how tasks are delegated. ORDIT (Dobson & Strens, 1994) and the approach proposed by Loucopoulos & Kavakli (1995) do include the

macro-organisational context. ORDIT focuses on the delegation of responsibilities but neglects structural organisational relationships. In contrast the approach of Loucopoulos & Kavakli focuses on deriving goals from the organisational activities, but does not clarify the lines of authority and delegation.

Recently researchers in requirements engineering (RE) have turned their attention to deriving access policies from requirements models, focusing on the assignment of tasks and resources to actors, and how they can be refined into access policies. Fontaine (2001) has explored the mapping of agent assignments in KAOS, a goal-based modelling framework for modelling goal hierarchies, to authorisation policies in Ponder – a language for specifying access control policies in distributed systems (Damianou *et al.*, 2000). He (2005) has proposed the Requirements-Based Access Control Analysis and Policy Specification (ReCAPS) method, a set of heuristics, to derive roles from task assignments in order to define RBAC policies, and Liu *et al.* (2003) have proposed how dependencies between actors, resources and tasks in the i* framework, can be used for defining RBAC policies.

However what this research has not demonstrated is how to relate actors to the organisational context; thus we still do not have a satisfactory way of deriving precise actor definitions, and although researchers have demonstrated a systematic approach to defining policies, they have not demonstrated that these policies satisfy the principles of management control.

## 1.2 Problem Description and Research Objectives

In order to define access policies satisfying the principles of management control, a prerequisite is that we can relate actors to the organisational context. An example of a management control principle is the minimum privileges principle. The principle of

minimum privileges constrains users to access only those resources that they need in order to be able to carry out their tasks (Anderson, 2001). Users can carry out similar functions but in different organisational units; e.g., bank clerks carry out the same function in local branches, but should only access accounts in the branch to which they are assigned; thus the organisational unit, in this case the branch, is a constraint in an access policy definition. This poses a challenge, especially as current requirements modelling approaches do not give us an explicit link to the organisational context. With respect to requirements modelling, van Lamsweerde (2000b) raises some key questions that researchers have been tackling:

- What aspects to model?
- How to model such aspects?
- How to define the model precisely?
- How to the reason about the model?

In this thesis, we are, in effect, pursuing a subset of those broad questions that van Lamsweerde (2000b) posed with respect to requirements in general, except that we are focusing on the organisational context, and access policies derived from the minimum privileges principle. We focus on the minimum privileges principle because of its fundamental nature, and other principles build on it.

Therefore, the key objectives of the research in this thesis are to:

1. define the organisational context and to relate it to actors;
2. define policies that satisfy the principles of management control, and in particular the minimum privileges principle;
3. verify scenarios are consistent with policies;
4. extend an existing requirements modelling approach, to relate actors to the organisational context, and define policies.

In this thesis we present a framework that comprises:

- a meta-model for defining the organisational context, and relating roles to the organisational context;
- heuristics for defining the organisational context, and deriving roles that relate to this organisational context;
- constructs for defining access policies satisfying the minimum privileges principle;
- constructs for defining scenarios;
- rules for verifying scenarios are consistent with policies.

The framework is actually independent of any specific requirements modelling approach, as it addresses the first three objectives, which are of a fundamental nature, in that they address what we need to model, and how it can be done independently of any given requirements modelling approach. The framework is defined formally in Z (Spivey, 1992) and gives us a meta-model, from which models of the organisational context for specific applications can be developed, and a set of constructs for formulating access policies and scenarios to verify these policies.

The fourth objective is to relate this framework to existing requirements modelling approaches. Rather than inventing a new language, it is sensible to extend an existing language, so that access policies can be modelled using the same language as other requirements. As described above there are a number of different approaches to modelling requirements. Of these approaches, the i* framework makes an ideal choice as it focuses on the social dependency of actors; organisations are in essence social systems. The i* framework uses a graphical language and is semi-formal; i.e. operationalised goals that refine into functions and constraints are defined in natural language. Recently, however, a variant of the i* framework has been developed,

formal Tropos, which allows requirements to be formally defined. Although this thesis focuses on extending the formal Tropos notation, we also propose how i* diagrams can be extended.

## 1.3 Research Method

Our approach to this research is to identify the key concepts required for modelling access policies drawing on the literature in requirements engineering, security policies, and organisational behaviour. The RE literature gives us the basis from which we can extend existing approaches; the security policy and organisational behaviour literature enable us to identify new concepts that can be used to extend existing approaches.

We validated the framework with regard to its value as an engineering approach empirically. There are at least three empirical research methods for validating research in RE (Sim *et al.*, 2003). The first is through experiments, the second is benchmarking, and the third is through case studies. Benchmarking and experiments have the advantage of allowing a direct comparison with other approaches using objective measures. However a case study approach is particularly appropriate in exploratory research, where the problem is not well understood or defined, as was the case of defining and analysing access policies in requirements models. Benchmarking and experiments are more appropriate in verifying theories that have already been well formulated. For this reason a case study approach was adopted. However, there is a danger in adopting a case study approach in that if it is tied to a specific situation, the findings can not be generalised. We have obviated this problem by selecting several case studies in diverse domains. We selected three case studies for exploration. The first case study, in the medical domain on the access policies for

medical records, was based on interviews we conducted with individuals who had experience of working in hospitals. The second case study, on access policies for resources in a software project, was based on the literature, and the author's experience within a software development organisation. We took the third case study, on access policies within a large European bank, from the literature. We used the third case study to verify the practicality of using an extended requirements modelling language, formal Tropos, for defining policies, and verifying them using a tool that automates the analysis.

## 1.4 Thesis Contribution

In this thesis we identify the need to model the macro-organisational context as a prerequisite for defining access policies, and furthermore that concepts already exist in the organisational and security policy literature that can be used as a basis for this (Crook *et al.*, 2002b).

We elaborate a framework that supports the modelling of access policies as requirements. The framework contains a meta-model that enables us to model key aspects of the organisational context, and from this to derive roles; the framework thus provides a link between roles and the organisational context. A set of heuristics provides a systematic way of deriving roles (Crook *et al.*, 2002a; Crook *et al.*, 2003).

The framework meta-model contains a construct that relates roles to tasks, which enables us to define access policies that satisfy the minimum privileges principle. In order to verify policies, the framework meta-model contains a set of constructs that enables us to define scenarios, and rules in the framework enable us to verify that the scenarios are consistent with the policies defined (Crook *et al.*, 2002a; Crook *et al.*, 2003).

The thesis demonstrates how the language constructs can be used to extend the requirements modelling language the i* framework and the formal equivalent of it, formal Tropos (Crook *et al.*, 2005), and how these constructs may be translated into the Alloy language and analysed using the Alloy model checker.

## 1.5 Thesis Road Map

The thesis is structured as follows:

Chapter 2 describes the RE context of the work; we discuss alternative approaches to modelling requirements. The complementary nature of the different paradigms is highlighted. In particular we focus on security requirements, and identify two principle sources, which are threats and the principles of management control. We review approaches to modelling security requirements. We identify key weaknesses of modelling requirements derived from the principles of management using existing approaches, in particular the lack of a link to the organisational context.

In chapter 3 we review the security literature on the principles of management control, and how access policies maybe defined to enforce them. We also review the organisational literature to understand the organisational context, in particular the rationale for organisational structures. We then revisit the problem of relating actors to the organisational context in current requirements models to define access policies.

In chapter 4 we review the i* framework, a requirements modelling language, and a formal version of it, formal Tropos, in depth. The i* framework models the social context, representing the dependencies between actors, tasks and resources. Using a case study we show how it can be used for defining access policies, and how the weakness identified in the previous chapter, in relating actors to the organisational context impacts on these access policy definitions.

Chapter 5 builds on the organisational characteristics elucidated in chapter 3; it addresses the second and third research objectives, proposing a framework for formally defining and refining access policies. The framework consists of a meta-model, which enables the organisational context to be represented and related to actors in order to define access policies, and a set of heuristics for expressing policies and scenarios. We define a set of rules for verifying scenarios from policies. The model is presented in the formal language Z.

Chapter 6 addresses the research objective in relating this framework to existing requirements modelling frameworks, and how it can be integrated. We demonstrate how the organisational meta-model presented in the previous chapter can be applied to extending formal Tropos, and i* framework diagrams. We then revisit the case study we explored in chapter 4 to show how the extended formal Tropos language can be applied to define access policies, and scenarios. We show how the access policies defined in formal Tropos can be mapped on to the Z based meta-model presented in the previous chapter.

Chapter 7 addresses the problem of how to verify policies using the framework. We propose translating the Z constructs of the framework meta-model into a specification language, Alloy, which is supported by a model checking tool. Using the tool, we demonstrate how scenarios can be checked against policy definitions.

Chapter 8 demonstrates through the use of a case study how the extended formal Tropos requirements modelling approach can be applied, and how a formal Tropos model representing access policies can be translated into Alloy and analysed.

Chapter 9 summarises the conclusions and contributions of the thesis, and sets an agenda for future work.

# Chapter 2

# Modelling of Security Requirements

In this chapter we describe the Requirements Engineering (RE) context of the work in this thesis. We examine existing alternative approaches to modelling requirements. We then identify what security requirements are, and highlight the importance of an organisational procedure as a special type of security requirement.

## 2.1 Modelling of Requirements

Nuseibeh & Easterbrook (2000) describe the core activities of RE as eliciting requirements, modelling and analysing requirements, communicating requirements, agreeing requirements, and evolving requirements. Within these activities, the modelling of requirements is central, as it supports the other activities and provides a basis for performing reasoned analysis, to validate requirements, to ensure consistency, and identify conflicts. There is a plethora of ways to model requirements.

### 2.1.1 Early Approaches to Requirements Modelling

In the 1970's and 1980's the emphasis was on the how and what of requirements (van Lamsweerde, 2000b); i.e. data modelling (the what) and data transformations (the how). Initially, semi-formal approaches based on data-flow and entity relationship diagrams were widely used. However such modelling techniques were

found to be inadequate due to limited structuring capabilities, and vague formulation using largely natural language (van Lamsweerde, 2000b). Subsequently formal specification languages came to the fore in the 1980's and early 1990's with languages such as VDM (Jones, 1990) and Z (Spivey, 1992); these mathematically based languages offered much richer structuring facilities, such as aggregation and instantiation, and allowed the expression of formal assertions. Formal languages are precise, and lend themselves to automated reasoning for detecting inconsistencies; they can also be used to validate specifications by animating them. A significant problem with regard to modelling requirements using these languages is that they do not separate the environment in which the system operates; i.e. the domain, and the description of the intentions of the system; these tend to be mixed up in a single specification (van Lamsweerde, 2000b). It is important to distinguish between the given problem domain and requirements (Jackson & Zave, 1997). The problem domain has "indicative" properties; i.e. properties of the environment that are given. Requirements are "optative" properties; i.e. they describe the system as it should be. This separation of concerns is necessary as the indicative properties of the problem domain represent constraints on how the system interacts with the environment, which can not be changed.

SCR is a specialised approach to requirements modelling, first proposed in the 1970's (Alspaugh *et al.*, 1992; Heitmeyer, 2002), it was the first to separate the intentions from the problem domain. It is a formal approach to modelling, enabling one to specify the behaviour of parallel finite state machines, and was developed for modelling and verifying high assurance process control systems. There exists a tool set for automating consistency verification (Heitmeyer, 1998). Although it is highly

suitable in modelling and simulating process control applications, it has limited capabilities in modelling the interaction with the environment (Zave, 1997).

In the 1990's researchers began to address the question of what to model, which lead to the inclusion of additional conceptual units such as agents, goals, and events in requirements modelling approaches, in addition to entities and functions (van Lamsweerde, 2000b).

### 2.1.2 Goal-Based Approaches to Requirements Modelling

Goal based reasoning has become an important thread in RE research, and forms the basis of a number of modelling approaches. Goals are objectives that the system needs to achieve, through the co-operation of agents in the system to be (van Lamsweerde, 2000b). Understanding why requirements are needed helps stakeholders and analysts to ensure that requirements are complete, and to evaluate the inevitable trade-offs that occur as a result of conflicting system objectives (van Lamsweerde, 2000b).

Various approaches to modelling goals and translating them into functional requirements have been proposed (Antón, 1996; Dardenne *et al.*, 1993). Top level objectives can be successively refined into lower level goals; at the lowest levels system requirements can be articulated. One framework for modelling goals is KAOS (Knowledge Acquisition in autOmated Specification) (Dardenne *et al.*, 1993). The framework comprises a language and a method for developing requirements models. A KAOS requirements model consists of a goal hierarchy. Low level goals are refinements of high level goals. At the lowest level are operationalised requirements, which are actions and constraints on those actions that fulfil low level goals. It

incorporates a first order temporal logic notation to express goals and operationalised requirements. The formal notation lends itself to automated checking.

Another approach to deriving requirements using a goal hierarchy is proposed by Antón (1996). Antón proposes a method, the Goal Based Requirements Analysis Method (GBRAMS), for goal based analysis. The resulting model from this method is a goal hierarchy. In this respect it is similar to a KAOS model. However it is expressed in natural language rather than formally. Antón focuses more on the heuristics for eliciting goals and refining them. Again as with KAOS, an identification of potential agents and assignment to actions is part of the process.

There is a category of requirements, known as non-functional requirements (NFR's), (Chung, 1991), such as performance, reliability, and also security. A different approach to defining and analysing non-functional requirements is necessary because, as Chung points out, they are often global constraints. The NFR framework (Mylopoulos *et al.*, 1992) offers a way for analysts and designers to explore how design decisions can contribute to or obstruct NFR goals.

### 2.1.3 The Organisational Context in Modelling Requirements

The goals of a system are often embedded in an organisational context; the organisational structures and business rules form the basis of the rationale from which goals are derived (Nuseibeh & Easterbrook, 2000). An approach which focuses on the organisation is ORDIT (Organisational Requirements Definition for Information Technology) (Dobson *et al.*, 1992; Strens & Dobson, 1994; Dobson & Strens, 1994). The basic premise behind this approach is that the organisational goals, policies, structures, and roles are essential to understanding organisational requirements from which a functional specification of the system can be derived. It includes a role

model, which captures functional and structural relationships, and responsibilities. The identification of responsibilities and how they are delegated are key because these lead to the questions as to how a system will support a user in executing those responsibilities.

Loucopoulos & Kavakli (1995) propose a similar approach. The essence here is to combine the technical and social perspectives. The social perspective is represented as a model of the organisational members, and how they interact. Central to this view is an actor that can be an organisational unit or individual, and can be assigned roles representing the responsibilities held by the actor. The technical perspective is represented by a model of the activities, data, and information flow. These perspectives are combined in an enterprise model, which also includes a goal hierarchy.

The i* framework (Yu & Mylopoulus, 1994) also focuses on the organisational context, modelling goals in the form of intentions of actors and dependencies between those actors. This focuses more on individuals, their intentions and dependencies on one another, than on how they relate to the organisational structure, and how responsibility is delegated; i.e. the micro-organisational factors. More recently the i* framework has become part of the Tropos methodology (Giorgini *et al.*, 2004). Tropos is a methodology for software development, supporting requirements analysis and software design. The i* framework is a semi-structured approach using diagrams and textual descriptions; recently a formal version of the i* framework, formal Tropos, has been proposed Fuxman *et al.* (2001). With formal Tropos goals, actions and constraints can be defined using the same first order predicate language with temporal constraints as used in KAOS.

### 2.1.4 Scenario-Based Requirements Modelling

Although the incorporation of goals into a requirements model is appropriate, they may be difficult to elicit. Stakeholders may have difficulty defining abstract goals but find it easier to describe operational scenarios. Cognitive studies on human problem solving have borne this out (Benner *et al.*, 1993), and in practice scenarios are widely used (Weidenhaupt *et al.*, 1998). A scenario is a temporal sequence of actions or events, describing the way agents interact with the system. Scenarios are useful for clarifying and validating abstract requirements models, such as a goal-based models (Antón, 1997; van Lamsweerde *et al.*, 1995). They also provide a suitable basis for identifying test cases.

One can differentiate between abstract scenarios, such as use cases and instance-based scenarios, which describe a specific situation (Maiden, 1998). Although there are advantages to conducting a scenario analysis there are also significant weaknesses (van Lamsweerde & Willemet, 1998). There is a problem in establishing whether a set of scenarios covers all the goals of the systems; i.e. they are inherently partial. It is difficult to identify conflicting goals; there is also a danger of explosion of scenarios with many different combinations of events and actions; the scenarios may be fragmentary without a clear link between them, even if one exists; and there is also the danger of specifying more details than are necessary in the way users interact with the system, imposing unnecessary constraints on the design (van Lamsweerde & Willemet, 1998).

Researchers have focused on how to combine the advantages of an abstract goal-based model and concrete scenarios to elicit, elaborate and validate requirements. Potts (1995) for example proposes a method to elicit salient scenarios. Before the scenario analysis is carried out, goals and obstacles to goals need to be identified. It is

from these that salient scenarios are defined; what Potts means by salient scenarios is that each one describes a unique combination of goals being achieved or obstructed; he describes a set of heuristics to do this but does not propose how to represent them. Van Lamsweerde & Willemet (1998) propose a formal method to derive goals from scenarios. The method exploits the formal language of the KAOS framework, and is a rigorous approach to deriving a set of goals from scenarios, as the consistency between the goals and scenarios can be proven.

Use cases, which are a part of the of the Unified Modelling Language, have become very popular in practice, and have been a significant driving factor in organisations adopting scenario-based approaches (Weidenhaupt *et al.*, 1998). Use cases are basically abstract scenarios describing all possible actions between the user and the machine and differ therefore from scenarios, which model specific sequences of actions (Maiden, 1998). Use cases are semi-formal in that text is used to describe the actions. In order to link goals with use cases, Cockburn (2001) recommends modelling high level use cases as goals, so in effect producing a similar model to those goal hierarchies we examined in the previous section. Formal approaches to abstract scenario analysis have been proposed using statecharts (Glinz,1995; Ryser & Glinz, 2001), message sequence charts (Uchitel *et al.*, 2001), and trees (Hsia *et al.*, 1994). These approaches obviate some of the weaknesses of a use case analysis, in that it is possible to link different scenarios, and their formal nature lends them to automated analysis.

### 2.1.5 Domain Analysis

A parallel thread of research is on the domain and the specification (Jackson & Zave, 1997). Jackson & Zave take the viewpoint that goals are not the appropriate

starting point for a requirements analysis; this is because high level goals can become divorced from the problem. They argue that the problem domain is the starting point; i.e. to understand how the machine, representing the software to be designed, interrelates with its environment. Based on this philosophy, Jackson (2001) proposes an approach to modelling using problem frames. The premise of this approach is that the problem; i.e. the way in which the machine interrelates with the environment, can be broken down into sub-problems each of which is easier to solve. A problem frame represents a template of a class of simple problems, for which a known solution already exists. Jackson identifies several basic frames. During the analysis phase the problem is decomposed into sub-problems according to their frame types. Each sub-problem is then analysed independently. Jackson argues that each type of problem frame requires its own specific analytical technique, and isolating each sub-problem in this way simplifies the analysis.

The idea that generic problem types exist and can be used as a basis for requirements analysis has also been identified by Sutcliffe & Maiden (1998). Sutcliffe & Maiden (1998), and subsequently Sutcliffe (2000) have built a library of generalised models that are applicable to different applications. For example the concept of object containment is applicable to both a library providing books, and a warehouse providing spare parts. As with problem frame analysis a key to this is the modelling of the problem domain. The advantage of this approach is the productivity gain in producing requirements specifications from reusable specification building blocks.

Although Zave & Jackson have suggested that goals are not an appropriate starting point, that is not to say that goals are inappropriate and that a domain analysis is better; as Sutcliffe & Maiden (1998) point out, a domain based requirements

analysis is complementary to approaches such as those based on goals, in that in addition to a domain analysis, goals and scenarios also need to be defined. Indeed, goal or scenario based models contain domain characteristics; the i* framework, for example, contains basic modelling elements for representing the organisational domain, and the KAOS requirements modelling language is based on a meta-model, which identifies fundamental elements applicable to any problem domain, such as actions, events, entities, and agents.

### 2.1.6 Summary of Modelling Approaches

Although there is a plethora of ways to model requirements, many of these approaches are complementary, an aspect on which researchers have often focused, such as the integration of goals and scenarios. Goals are important to ensure the completeness of requirements. Scenarios help elicit, elaborate, and validate requirements. Modelling business processes and the organisational context provides the rationale for goals. The modelling of the problem domain provides us with the building blocks for formulating requirements without presuming a solution. The formalisation of a model enables us to automate checking to verify consistency and identify conflicts. Research in RE has focused mainly on functional requirements but is applicable to security requirements, and it is within this context that we now explore security requirements in more detail. Table 2.1 overleaf summarises the approaches we've reviewed.

| Paradigm | Modelling Approach | Summary of Approach |
|---|---|---|
| **Goal Orientation** | KAOS (Dardenne *et al.*, 1993) | Goals successively decomposed into operationalised requirements supported by a formal notation. |
| | GBRAMS (Antón, 1996) | Method for eliciting and elaborating requirements from a goal hierarchy, using natural language to describe functions. |
| | NFR Framework (Chung, 1991) | Semi-formal approach to elaborate non-functional requirements from goals, linking design decisions to goals. |
| **Organisational Context** | i* Framework (Yu & Mylopoulus, 1994) | Requirements derived from the strategic intentions of actors and their dependencies. A structured approach with diagrams and textual descriptions. |
| | Formal Tropos (Fuxman *et al.*, 2001) | Formal version of the i* framework. |
| | ORDIT (Dobson *et al.*, 1992) | Requirements derived from the responsibilities of users, semi-formal based on diagrams and textual descriptions. |
| | Enterprise Modelling (Loucopoulus & Kavakli, 1995) | This semi-formal approach combines the social perspective (actors, their roles and goals) with the technical perspective (flow of information and business processes) to derive requirements. |

| | | |
|---|---|---|
| **Scenarios** | Use Cases | Semi-formal approach to describe scenarios. |
| | Statecharts (Glinz, 1995) | Formal approach to describe scenarios using statecharts. |
| | Message Sequence Charts (Uchitel *et al.*, 2001) | Formal approach to describe scenarios using message sequence charts. |
| | Trees (Hsia *et al.*, 1994) | Formal approach to describe scenarios using trees. |
| | Schematic Scenario Analysis (Potts, 1995) | Structured method to derive goals from scenarios based on textual descriptions. |
| | CREWS (Maiden, 1998) | Semi-formal approach to explore alternative scenarios generated from an abstract model. |
| | Inferring Declarative Requirements from Scenarios (van Lamsweerde and Willemet, 1998) | Formal approach to identify goals from scenarios. |
| **Domain** | Problem Frames (Jackson, 2001) | Requirements derived by decomposing problems according to generic types of problems. |
| | Domain Matching (Sutcliffe, 2000) | This approach identifies and matches requirements to generic types. |
| **Finite State machine** | Software Cost Reduction (Heitmeyer, 1998) | Models the behaviour of a system as a set of outputs expressed as a mathematical function of the state and history of the environment. |

**Table 2.1 Overview of Requirements Modelling Approaches**

## 2.2 Security Requirements

In this section we explore what security requirements are and highlight the significance of security requirements derived from the principles of management control.

### 2.2.1 The Source of Security Goals and Requirements

Security requirements are those requirements concerned with the protection of valuable assets. They stem from the top level security objectives of maintaining confidentiality, integrity and availability (ISO, 2005). These top level objectives give us an orientation. Confidentiality is concerned with maintaining privacy and secrecy, allowing read access to only those users who have been authorised. Integrity is about ensuring the accuracy and completeness of information, and involves allowing only authorised users to change or create data and applying controls to ensure the correctness of the data. Availability is concerned with ensuring that access to information systems is maintained when required. Closely related to security requirements are privacy requirements. Privacy requirements differ from security requirements in that they are associated with the protection of personal data, rather than data belonging to an organisation.

These high level security and privacy objectives express what we want to protect with regard to valuable assets, but need to be translated into security goals, and subsequently security requirements. Moffett *et al.* (2004) provide a perspective on the source of security goals and how they relate to security requirements. A goal is something that a stakeholder wishes to achieve or avoid. A security requirement is a constraint on a function required to achieve a security goal. Moffett *et al.* identify three sources of security goals:

1. Harm to assets
2. Management control principles
3. Business goals

Harm to assets can occur when the system is misused, and a security objective is breached, such as the confidentiality or integrity of the asset, hence specific goals are derived to obviate these threats. Constraints also stem from the principles of management control that apply to all applications, and would otherwise be repeatedly derived. The third source are the business goals for specific applications, which determine what assets are at risk, and which principles of management control apply. Figure 2.1 illustrates these sources. Moffett *et al.* (2004) discuss policies, but restrict their discussion to access policies. Access policies are however derived from organisational requirements, as pointed out by Thomas & Sandhu (1994); Moffett *et al.* do not explain this derivation process.

Antón & Earp (2001) emphasise the need to align security and privacy requirements with a security policy. They describe a security policy as a document identifying security goals and assessing risks. Antón *et al.* (2001) use the term of meta-requirement to describe a policy. By this they mean that a policy is a requirement applicable to all systems within an organisation. Antón & Earp (2001), and subsequently Antón *et al.* (2001) propose the use of GBRAMS to systematically define security goals that constitute a security policy, and to ensure that security requirements in a system are aligned with the security policy. Unlike Moffett *et al.* they ignore the principles of management control.

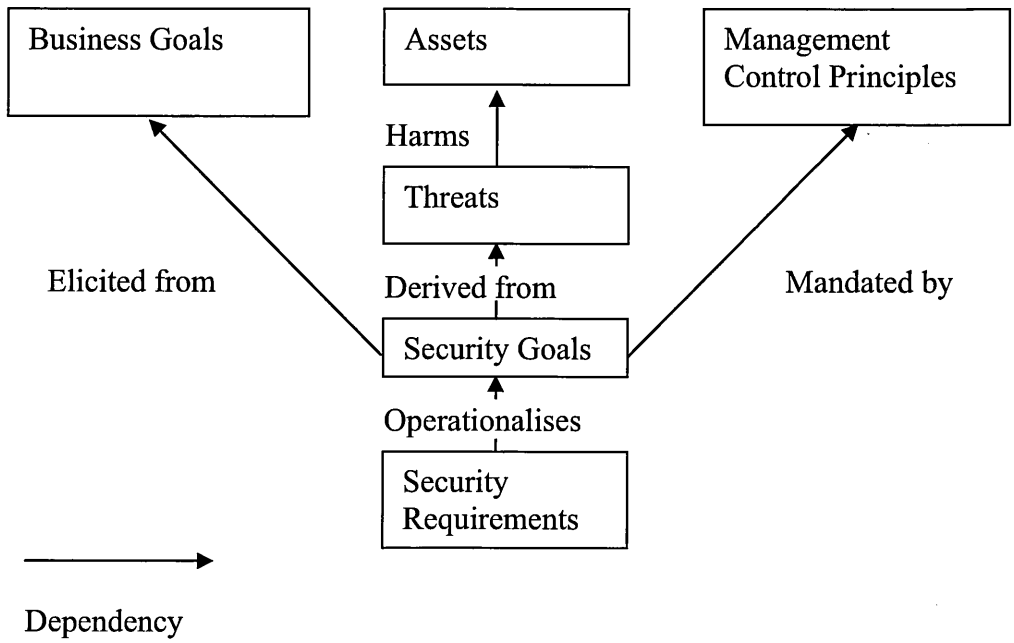



**Figure 2.1 Security Requirements Core Artefacts (adapted from Moffett *et al.*, 2004)**

Policies are expressed in natural language, and Breaux & Antón (2005) suggest they are more open to interpretation than goals expressed in a requirements specification. They have proposed a process to derive semantic models from goals extracted from privacy policy documents, which enables policies and goals to be compared more easily. This research builds on the approach proposed by Antón *et al.* but focuses on privacy policies rather than security, and hence does not explore policies derived from the principles of management control.

### 2.2.2 Deriving Security Requirements from Threats

Researchers have explored how some of the different modelling approaches that we reviewed in section 2.1 can be adapted to analyse threats, and so derive security requirements.

Sindre & Opdahl (2000), and Sindre & Opdahl (2001) propose an approach to modelling threats, based on use cases, which they call "misuse cases". A conventional use case diagram is extended by adding inverse cases, which model how malicious actors may perform harmful actions; having identified these threats, countermeasures can be defined. It is a systematic approach to modelling threats, though not formal, and analysis is subjective. It does however make the analyst think about each use case, and whether there is a scenario which could be harmful.

Alexander (2002) builds on this approach, but focuses more on the conflict resolution of goals. Alternative use case goals may mitigate or aggravate the threats posed by misuse cases, and Alexander proposes a notation to indicate this on use case diagrams. Although the resulting model shows where goals conflict, what this approach does not show is how goals aggravate or mitigate threats.

Van Lamsweerde *et al.* (2003) and van Lamsweerde (2004) propose KAOS to model threats and countermeasures. They model threats as "anti-goals", representing the malicious intent of agents as obstacles to security goals. Having discovered threats, further goals can then be defined to counteract the threats. This is a formal approach, which enables a more rigorous analysis to be carried out than with use cases.

Yu & Liu (2000) and Liu *et al.* (2003) show how a threats analysis can be carried out using the i* framework. The i* framework models the social intentions of actors. The authors show how actors can be modelled in attacking roles, and how attacks, modelled as goals, and their task dependencies, impact on the security goals of other actors. As with the other approaches, having identified attacks, countermeasure goals can then be added, in order to mitigate attacks. This is a very similar approach to the one proposed by Alexander, though the i* framework allows the definition of a more

detailed model, where individual actors can be assigned multiple roles, and soft goals, tasks, resources, and their dependencies can also be modelled.

Lin *et al.* (2003) apply problem frames to bound the scope of a security problem. Their approach is to first of all scope the problem, identifying sub-problems. The next step is then to identify the security concerns of each sub-problem. Threats to the security concerns, which the authors refer to as "anti-requirements", are then captured in the form of "abuse frames". An abuse frame is a problem frame that models the threat to the system context showing the phenomena by which malicious users interact with the system. This analysis highlights vulnerabilities, which then can be used as a basis for improving the design. One of the key advantages of problem frame analysis is that recurring types of problems can be identified and described, however Lin *et al.* do not show how this would be applied or how it is advantageous for the analysis of abuse frames. They point out that some threats only become evident by recomposing sub-problems.

Haley *et al.* (2004) have explored how trust assumptions can be considered during an analysis. A trust assumption is a security property that a component, be it human or technical, must posses if a security requirement is to be satisfied. They illustrate using problem frames how trust assumptions can be assigned to components within the problem domain. Structured argumentation can be applied to verify that trust assumptions have been satisfied (Haley *et al.*, 2005). It complements the research by Lin *et al.* (2004), focusing more on the design constraints that need to be satisfied to ensure that security requirements can be fulfilled.

All these approaches are systematic in their identification of possible attacks and the definition of countermeasures. They all involve an iterative process of defining security goals and identifying ways by which these goals can be obstructed or broken.

However the process of identifying threats is still a creative process; it requires experience of what has happened in the past to invent scenarios. Unlike functional requirements, where stakeholders are on hand to express what they want from a system, malicious users are not on hand to express how they intend to attack the system. This is why it would be helpful to have a way of identifying recurring threats. This approach has been alluded to by Lin *et al.* (2004), but is an avenue of research that still needs to be pursued.

### 2.2.3 Deriving Security Requirements from Management Control Principles

In commercial organisations, there are established management control principles, which are applied to protect assets and prevent fraud (Moffett & Sloman, 1988). Accounting practices are a key source of these principles as Clark & Wilson (1987) have identified. For example double entry book-keeping entails that any transaction has two parts in two separate ledgers, where a transaction booked into one ledger is matched by a transaction booked out of another ledger. If a transaction in one book is not matched, as is checked during an audit, then this is either an error or fraud. The segregation of duties provides a further mechanism in that if it is ensured that matching transactions can not be entered by one individual, then fraud is less likely, as to commit fraud, collusion is necessary.

In fact the cause of the fraud at Barings Bank that we described in chapter 1, was a breakdown in the accounting procedures and segregation of duties. The perpetrator, Nick Leeson, was in charge of two separate areas in the bank, settlement and trading, which enabled him to hide unauthorised trades and so manipulate the books. It does illustrate how important it is to maintain these principles.

There are other key principles from which access policies are derived. The minimum privileges principle requires users only have access to those resources that they need in order to be able to carry out their tasks (Anderson, 2001). Moffett & Lupu (1999) identify the delegation and revocation of authority, and supervision and review as two further principles.

Thomas & Sandhu (1994) view an organisation as a system that is required to preserve a certain level of integrity, and that there are organisational procedures and internal controls required to maintain this state. This integrity has also to be maintained in the organisation's computer systems. They highlight that control principles are organisational requirements that ultimately translate into access control models to enforce these principles.

The way in which such controls can mitigate threats is illustrated by Anderson (1996). In describing a policy for medical records he highlights the fact that there would be much greater concern if several thousand GP receptionists could all access the medical records of any patient in the UK, than if each one could only access the records of patients in the practice in which they work. This is a good example of how the minimum privileges principle can be applied to mitigate the threat to the confidentiality of patient records.

The fact that management control principles are a key source of security goals and requirements means that we need to be able formulate goals and requirements that do satisfy these control principles. To a great extent they will be formulated as access policies. Recently research has started to focus on how requirements modelling approaches can be applied to defining them.

Fontaine (2001) has explored the mapping of agent assignment in KAOS to access policies in Ponder, a language for specifying access control policies in

distributed systems. A significant problem here is that Fontaine does not differentiate between actual users or roles; i.e. that a physical individual can adopt several roles. An example where this is a problem is if we wish to define the segregation of duties, whereby we are interested in preventing a physical individual from performing two separate tasks. Fontaine does highlight the agent, and the agent's assignment to actions as a key to the definition of access policies.

Bandara *et al.* (2004) have also explored deriving policies in Ponder from a KAOS goal model, however they focus specifically on the enforcement of sequences of operations to satisfy goals rather than agent assignments.

Liu *et al.* (2003) have explored how the minimum privileges principle can be modelled in the i* framework, using a strategic rationale (SR) model, which models the internal relationships within an actor with respect to the goals, tasks and resources; the goals, resources and tasks are contained within an actor boundary. They show how the actor boundary in the SR model can be used to define access restrictions, an example of which is shown in figure 2.2 adapted from their case study. They also demonstrate how the minimum privileges principle and segregation of duties policies can be translated from i* into the specification language Alloy, which can then be checked automatically using a tool.

The actor boundary in figure 2.2 shows the relationship between the family doctor and the internal actor goal of providing a regular clinical service, which is dependent on the task to open a new medical record, which in turn is dependent on the resource, medical record. Liu *et al.* propose that the actor boundary defines restrictions ensuring the principle of least privileges can be enforced. The family doctor is then restricted to the tasks and resources defined within its boundary. However there is an important aspect associated with the least privileges principle that can not easily be represented

in i*; that is that the family doctor should only access medical records associated with his patients, otherwise he could access records associated with patients not assigned to him so violating the least privileges principle.

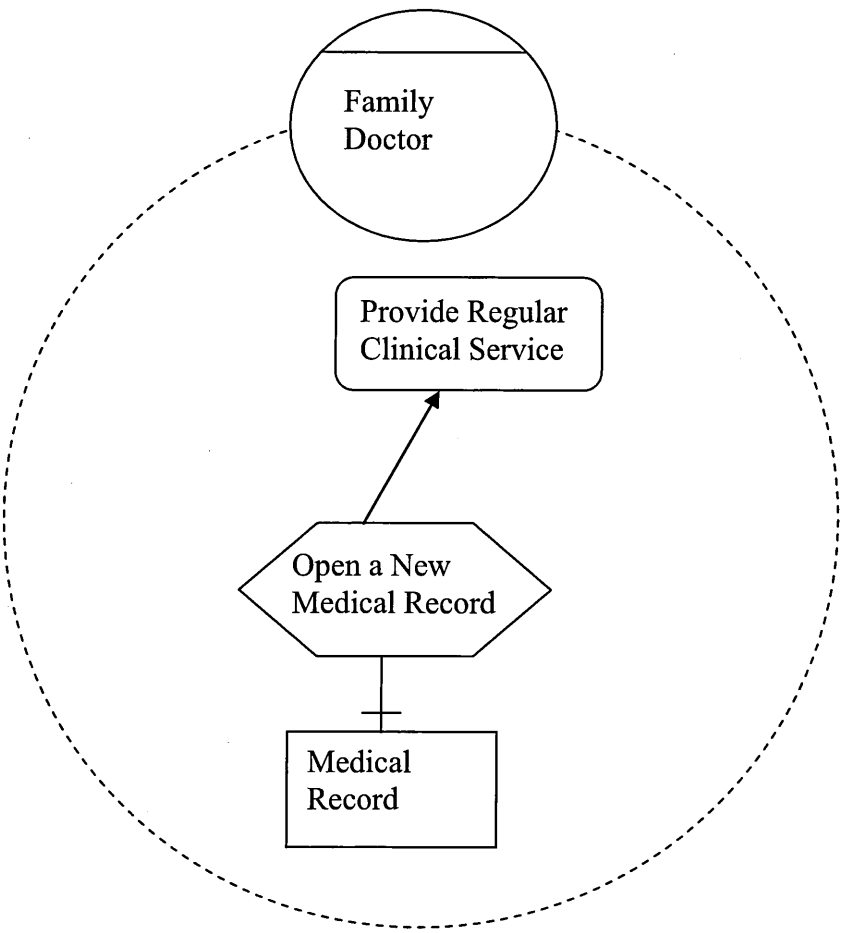



**Figure 2.2 Strategic Rationale Diagram for a Medical Application (adapted from Liu *et al.*, 2003)**

Giorgini *et al.* (2005) demonstrate the modelling of delegation and trust in i* diagrams. The authors propose extensions to i* dependency diagrams, within the context of the Tropos methodology Giorgini *et al.* (2004), to represent trust between agents and delegation relationships. Their focus is in relating trust and delegation; i.e. identifying to what extent the delegator can trust the delegatee. Their definition of delegation is more general than that derived from the principles of management

control, including delegation between agents not in the same organisation, such as a customer delegating to an organisation. Their approach however does not include identifying authority relationships, within an organisational domain. The importance of defining authority relationships is illustrated by the fraud at Barings Bank that we have already touched on; Nick Leeson abused his authority by delegating fraudulent tasks to his subordinates. Massaci & Zannone (2006) apply the approach proposed by Giorgini *et al.* to analyse a case study on the fraud committed by John Rusnak at the Allied Irish Bank. The key strength of this analysis is in detecting inconsistencies with regard to trust. However although they show the organisational context, there are certain key aspects that are not represented, for example the authority relationship between John Rusnak and his superior, and what responsibility he had.

He (2005) has proposed the Requirements-Based Access Control Analysis and Policy Specification (ReCAPS) method, which is a role engineering process for deriving roles that can be mapped onto an access control system, whereby a role is defined as a collection of permissions. It involves analysing tasks and the resources that need to be accessed as a result of carrying out these tasks, and defining roles as collections of permissions. A key strength of this approach is that it provides a systematic way of defining roles, and it links these definitions back to security goals. However, he does not explain how his approach relates to management control principles and how these principles maybe satisfied. Nevertheless the fact that roles are defined, which restrict access, gives us requirements satisfying the minimum privileges principle, but as with the other approaches described above, this would not be completely covered. He *et al.* (2006) present a case study using the ReCAPS method, where they give a role definition of an analyst in a software project, who can classify goals. But what they do not define is the scope; i.e. does an analyst classify

goals for one project, or perhaps he can classify goals for all projects carried out by an IT development department? As we saw above for the medical record policy proposed by Anderson (1996), this domain oriented restriction is important.

Strens & Dobson (1993) propose how ORDIT can be applied to derive security requirements. The ORDIT modelling approach allows authority relationships between roles to be defined. As with the i* framework the authors make an explicit delineation between human users and the roles that they can adopt. In particular they focus on defining responsibilities; they differentiate between responsibilities and obligations. They describe a responsibility as a state of affairs, whereas an obligation is what an agent needs to do to discharge his responsibilities; i.e. the expectation of carrying out activities. Whereas obligations can be delegated, responsibility remains with the delegator, and hence there is a relationship between two agents, where one agent has a responsibility to ensure that a task is carried out and the second agent, the subordinate, executes the task. This modelling technique uses diagrams to describe roles, and their relationships. Its strength is in the modelling of delegation; however, the approach although structured is not underpinned by a formal model in the same way that KAOS or the i* framework is, and hence the semantics are imprecise, and no details are given as to how delegation can be reasoned about. Although the hierarchical relationships can be modelled, there is no link to the organisational structure, and hence definitions relating roles to organisational domains are missing.

Sutcliffe *et al.* (2006) apply domain matching and problem frame analysis to a telemedicine application, which includes access control, but they only focus on the access control mechanisms rather than the assignment of tasks to agents; i.e. how users are checked against authorisation lists, rather than which users should be given access.

When we consider these contributions to modelling security requirements derived from management control principles, then we see that most of these principles have been explored, implicitly or explicitly, namely the principle of minimum privileges, segregation of duties, delegation, and supervision and review. But there still remains a problem, and that is the lack of a link to the organisational structure, organisational domains, and organisational functions. For example a restriction that a doctor can only access medical records of his patients, or a ward secretary register patients on a ward relate to the way that these obligations are delegated within the organisation, and these are not represented in the models that we reviewed above. As we have seen these restriction definitions are therefore incomplete in that they don't include important aspects of the organisational context.

The principle of delegation and revocation of authority, as well as that of supervision and review are further principles that require an analysis of the organisational context as they depend on the hierarchical relationships within the organisation (Moffett & Lupu, 1999). There are also accounting principles, such as credit limits and double entry book keeping. Thomas & Sandhu (1994) illustrate how procedures can be modelled in the form of workflows, where actions need to be carried out in a predefined sequence, with certain actions requiring approval.

For the purposes of comparison of those approaches for which the definition of access policies has been explored, table 2.2 summarises the extent to which requirements have been derived from key management control principles. As we see from table 2.2, two principles have been explored as to how they can be achieved using existing approaches by defining access policies as the assignment of tasks and resources to actors, but with only partial success. For modelling all these principles the key is to link actors not only to tasks and resources, but also to the organisational

context, and thereby lies the crux of the problem, that current modelling approaches do not easily allow this.

| Management Control Principle | KAOS | i* Framework | ReCAPS | ORDIT |
|---|---|---|---|---|
| Minimum Privileges Principle | partially | partially | partially | partially |
| Segregation of Duties | No | partially | no | no |
| Delegation and Revocation of Authority | No | partially | no | partially |
| Supervision and Review | No | no | no | partially |
| Accounting Principles | No | no | no | no |

**Table 2.2 Coverage of Management Control Principles by Requirements Modelling Approaches**

| Modelling of the Organisational Context | KAOS | i* Framework | ReCAPS | ORDIT |
|---|---|---|---|---|
| Agent Assignments to Tasks and Resources | yes | yes | yes | yes |
| Separation of Roles to Agents | partially | yes | no | yes |
| Organisational Domains | no | no | no | no |
| Organisational Functions | no | no | no | no |
| Authority Relationships | no | no | no | yes |
| Workflows | no | no | no | no |

**Table 2.3 Modelling of the Organisational Context in Requirements Modelling Approaches**

Table 2.3 demonstrates key characteristics of the organisational context that are important in modelling organisational principles. As we see in table 2.3, agent assignments to tasks and resources already exist in current models, and it this aspect that researchers have begun to explore to define the minimum privileges principle. The separation of roles and agents is necessary to model and analyse the segregation of duties. Organisational domains are required to ensure that we can define minimum privileges, and also determine the boundary within which authority can be exercised. The functions of the organisation determine which tasks need to be carried out. Authority relationships need to be represented as a prerequisite for modelling the principles of delegation, and supervision and review. Workflows are required to ensure that procedures can be modelled involving review, approval or satisfying accounting principles, such as only issuing cash orders for payment after the goods have been received.

To develop a framework to model the organisational context is the prime objective of this thesis, as this is a prerequisite for defining the security requirements that are derived from management control principles. We have selected the minimum privileges principle for investigation to demonstrate how our framework can improve on existing requirements modelling approaches. We have decided to extend the i* framework due to its focus on the social context. Organisations are social systems, and the i* framework's capabilities with respect to modelling the dependencies between actors can be applied to this context.

## 2.3 Chapter Summary and Evaluation

In this chapter we began with a general review of requirements modelling; although there is a plethora of ways to model requirements, there are some basic

paradigms common to these different approaches. The key aspects around which these paradigms are based include scenarios, goals, the organisational context, and the problem domain. These paradigms are not so much alternatives as complementary ways of viewing requirements, and indeed there has been research in to how to combine the strengths of different approaches; notable are the links between the organisational context and goals, as well as the complementary nature between abstract models, in particular goal-based models, and instantiated scenarios.

The chapter highlighted the nature of security requirements as being concerned with the protection of information assets. We identified three key sources of security requirements: the first source is from the analysis of threats; the second is from the business goals; the third being the principles of management control.

With respect to analysing threats to assets derived from business goals, researchers have explored a variety of existing modelling approaches. Threats are relatively straightforward to represent on models as obstacles to goals or anti-requirements; more difficult is the creative process of identifying how a system can be compromised. A challenge that researchers are now facing is how to identify recurring threats, which would help automate the process.

Management control principles are the third key source, and this chapter highlighted that recent research into deriving security requirements from management control principles has focused on defining access restrictions to satisfy the minimum privileges principle.

This chapter has specifically motivated the need to define access policies that satisfy the principles of management control during requirements modelling. Various principles were identified including the minimum privileges principle, segregation of duties, delegation and revocation of authority, supervision and review, and accounting

principles. The principle of minimum privileges has been selected for investigation as to how access policies can be defined to enforce them. The reason this principle has been selected is that it is fundamental, and other principles largely build on it; e.g., financial controlling and accounting principles. Furthermore the focus of the research is on, what the organisational literature refer to as, bureaucratic organisations, with role cultures, as it is organisations of this type that rely on formally defined procedures.

# Chapter 3

# The Organisational Context and Access Policies

In the last chapter we identified from the literature that there are security goals, and that they are derived from generic principles by which management control the actions of their subordinates. These security goals are satisfied by access policies that enforce these principles, for example, by ensuring that users only have access to resources they require to carry out the tasks delegated to them. Hence an access policy is a requirement that is derived from the principles of management control. In this chapter we explore this further. Firstly we review the organisational literature, which enables us to understand the principles by which organisations are structured, and how work is assigned within the organisation. Then we turn our attention to the security literature from which we gain insights into how access policies can be defined that satisfy these principles. What is of particular interest is how roles can be used as a basis for defining access policies, and how roles can be used to represent the organisational context.

This is very relevant to requirements modelling. As we saw in the last chapter, an access policy can be modelled from the assignment of an actor to tasks and resources, and an actor is in effect a role. We need to establish for each function in each system, who has access, and if we want to make sure that the policy is enforced, it entails an understanding of the organisation, and how tasks are assigned. Similarly, in order to

ensure that supervision and review are adequately enforced, we need to understand the lines of authority within the organisation; i.e. who is in charge of who.

## 3.1 The Theory Underlying the Organisational Context

In order to gain a deeper understanding of the organisational context, it is useful to explore the organisational literature to understand the rationale of organisations, organisational structure, and the co-ordination mechanisms. It complements the security literature as it delves deeper into the nature of organisations.

### 3.1.1 Organisations and their Purpose

Groups of people can achieve far more than individuals working alone. Organisations can be considered as "social arrangements for the controlled performance of collective goals" (Buchanan & Huczinsky, 1985). Organisations are social systems with purposes as wide ranging from baby sitting circles to multi-national chemical manufacturers. They vary widely from informal organisations, such as entrepreneurial start-ups to formal organisations, such as banks and government services.

The dilemma that organisations face is that the goals of individuals in an organisation can differ from the collective purpose of the organisation, such as when an individual commits fraud at the expense of the organisation. It is therefore necessary for organisations to exert control; this is the reason why organisations have a deliberate and ordered allocation of functions, and control the activities and interaction between organisational members. It is precisely these mechanisms that organisations also use to mitigate the actions of malicious employees.

The organisational structure, which is the fundament of management control, includes the allocation of formal responsibilities to interrelated groups and roles, it

also includes linking mechanisms between roles and the co-ordinating structures of the organisation. A formal hierarchy of command is usually represented by an organisational chart.

### 3.1.2 Organisational Structures

In particular, large organisations are composed of organisational units that have clearly defined spheres of competence, however there are different ways in which spheres of competence can be allocated (Handy, 1985). Two principle ways in which organisations are divided, are horizontally and vertically, referred to as horizontal and vertical differentiation respectively.

Vertical differentiation refers to the division of work between management levels, with respect to the administrative tasks such as planning, co-ordination and control across different functional areas, whereas horizontal differentiation refers to the division of work according to factors such as function, geography, or personal qualification (Handy, 1985).

However considering horizontal differentiation, Mintzberg (1978) suggests that although there are a number of factors that have been identified on which horizontal differentiation can be based, they can be categorised into functional and market based characteristics.

**Functional characteristics**: In addition to the division of work on the basis of function, Mintzberg has identified qualification and work process as ways of dividing work, which are essentially a special form of function oriented differentiation.

**Market characteristics**: The other way of fundamentally dividing work is on the basis of market based characteristics. This can include organisational division based on customers, service, product, location, or time. In this case the functions are

replicated but the difference being the market that the organisational division or unit is responsible for.

Often, particularly in large organisations, several of these characteristics are used. The National Health Service in the UK is divided into regional health authorities that, in turn, are composed of hospitals to serve the different population centres, so that the health authorities and hospitals are organised on a geographical basis. A hospital, however, is organised on a functional basis according to administration, the medical specialities, and supporting services. Similarly, retail banks have autonomous branches dispersed to serve local markets with a functional structure in each branch.

Another way in which an organisation can be structured according to multiple factors is through a matrix structure. In this case, each member of the organisation will belong to two groups. One group is responsible for the product or market, and the other has a functional responsibility. An example of this is in an engineering company undertaking projects. Each project consists of a multidisciplinary team of engineers, and each member of the team reports to the project manager, but there are also departments that carry responsibility for staff development, and maintaining standards in the different engineering disciplines.

This insight by Mintzberg has significant implications in modelling the requirements of access policies. We see this separation of functions and markets incorporated into security policies and frameworks. For example the principle of Chinese walls in financial institutions is based on preventing users from accessing data from clients who compete. Consultants are assigned sets of clients that represent markets, but carry out the same functions within their designated market segments. For the least privileges principle we need to take into account not only the functions that users can access, but also the market in which they operate. An example of this is

given by Schaad (2003) whereby bank clerks in each branch of the bank would carry out identical functions but only for customers served by the particular branch to which they were assigned.

### 3.1.3 Organisational Control Principles: A Perspective from the Organisational Literature

To what extent can we generalise about the way in which a company exercises control? There is a stark difference between a small entrepreneurial company and a multi-national bank. Within any organisation there is a common set of beliefs and values with regard to the way in which authority is exercised; this is termed the culture (Handy, 1985).

Roger Harrison's view on cultures provides a framework for analysis (Handy, 1985). Many large organisations such as banks have a role culture, exemplified by control through rules and procedures, where a role or position is more important than the individual; in contrast small organisations usually have a power culture. The co-ordinating mechanisms in a power culture are informal, and the organisation is flexible; there is little if anything in the way of formal procedures, and individuals are more important than roles. Another type of culture, found in organisations that run projects, is a task based culture. A matrix organisation is an example, where groups of specialists are formed to perform a particular task. Influence here is based on expert power rather than positional or personal power. Examples of this kind of culture can be found in IT development and investment banks.

These cultures are not mutually exclusive, as often different parts of an organisation will have different cultures. The strategic apex company may have a power culture, whereas the operational core could have a role culture, and in addition

there is a continuum between a power and a role culture. As companies grow there are increasing pressures to formalise organisational structures, and introduce more rules and procedures.

Mintzberg (1992) also classifies organisations in a similar way, making a key distinction between simple informal structures common in small organisations and larger formalised bureaucracies. He also identifies the key co-ordinating mechanisms in the different types of organisations. In simple structures co-ordination is achieved through what Mintzberg terms as "mutual adjustment"; i.e. informal communication; as an organisation grows, it then begins to rely increasingly on supervision, and in large organisations, rules and procedures become important.

Thus the principles of management control that we described in the previous section apply much more to large organisations, with a role culture, than small entrepreneurial organisations, however although there are classifications, it is not clear cut with a continuum between these cultures.

The focus in this thesis is on large organisations; i.e. a role culture, and how computer systems can support the enforcement of rules and procedures.

## 3.2 Modelling Access Policies: A Security Perspective

In chapter 1, we highlighted two key research questions pursued in requirements engineering research (Lamsweerde, 2000b), which are what aspects to model, and how to model them. With respect to access policies, it is something researchers in security engineering have been exploring for many years. Their main focus has been in the development of policy languages and access control models having more to do with the implementation of security than the analysis of security requirements. However these access control models contain more than technical mechanisms, such

as encryption, policy enforcement, and secure protocols, which in a requirements model only interest us as design decisions to satisfy a requirement. In developing policy languages, security researchers have found ways to express rules to restrict users, which ultimately need to be reflected in a requirements model. It is therefore relevant to review these languages and models. There are many policy languages and models, and it is not the objective of this thesis to provide a comprehensive review of them all; we focus on only those research contributions that have explored the organisational context, and show how they have successfully been able to model aspects of the organisational context.

### 3.2.1 Groups in the Definition of Access Policies

A common way of defining access policies is through the use of groups. Users have access to resources based on their membership of a group. Policy languages such as Ponder (Damianou *et al.*, 2000) and ASL (Jajodia *et al.*, 1997) allow us to define authorisations in this way; for example, the following is a language statement in Ponder:

**+auth** expermental_drugs

**subject** /clinicians/consultants

**action** prescribe_experimental_drugs

**targets** /patients

It defines an action on a target domain on which the action can be carried out, and a subject; i.e. the group of users that are authorised to carry out an action. It describes a policy that states, consultant clinicians can only carry out the action to prescribe experimental drugs to patients. The plus sign indicates that this is a positive authorisation policy that allows access; in contrast, a negative policy denies access.

This policy uses a subject domain. These subject domains are effectively instances that can be used to represent organisational groups, in the form of a hierarchy. Lupu *et al.* (2000) demonstrate this in a case study for a GSM mobile telecommunications enterprise, where the different organisational groups are defined as subject domains. In this geographically dispersed organisation, a subject domain is defined for each region, and the branches are modelled as sub-domains of their respective region. This reflects a division of work based on location, a form of division of work identified by Mintzberg that we reviewed previously.

Subject domains or groups are useful for representing policies on instances, and although it is appropriate to define instances of organisational groups in an implemented distributed policy management system, it is not useful in a requirements model. In a requirements model we do not really want to model hundreds of branches of a large organisation. It would be much more efficient to define an abstraction of a branch, which maps onto hundreds of branches.

### 3.2.2 Roles in the Definition of Access Policies

A way in which we can achieve this desired abstraction is through roles. Researchers in security have turned their attention to roles. Role-Based access control (RBAC) originally emerged from the software industry (Sandhu *et al.*, 1996), as a convenient way for administrators to define access constraints. A role is a collection of permissions; it differs from a group, which is a collection of users. Although roles can be defined to reflect membership of a group (Sandhu *et al.*, 2000), where a group is a set of instances of users, this focus on permissions rather than users gives us an ability to abstract about users and their access rights.

Thus, in the previous example the consultant immunologist could be defined as a role instead of a subject, and then it is applicable throughout every hospital, as the role can then be instantiated for a specific subject and target domain. This principle is used in the Ponder policy language (Damianou *et al.*, 2000). In Ponder a role is used to represent a position in an organisation (Lupu *et al.*, 2000). A role is not tied to a specific group of users, it must be instantiated. A role definition such as this is therefore useful in a requirements model, though we somehow need to specify how it may be instantiated.

Roles give us a useful abstraction; but there is an interesting question as to what a role actually represents, and how roles relate to one another. In this thesis we are interested in representing roles in requirements models as a way of providing us with abstract policy definitions, it is therefore useful to review the security and organisational literature on roles, as it helps us to understand what we expect from a role definition.

Sandhu *et al.* (1996) propose relating roles in an inheritance hierarchy, their suggestion being that senior roles inherit permissions from junior roles; in this way an inheritance hierarchy could represent the organisational hierarchy. However the use of inheritance in this way will often be undesirable, as recognised by the authors themselves. A project manager, for example, may not have sufficient expertise to carry out specialised tasks that his subordinates have been assigned, hence it would not be desirable for him to be assigned these permissions.

Moffett (1998) describes however how an inheritance hierarchy can be useful. It is often possible to identify common responsibilities amongst members of an organisation. These responsibilities can be bundled together to form a generalised role. An example of this in a hospital, say, would be to define a generalised role of

health care provider that provides permissions shared by both doctors and nurses. The role doctor is itself a generalised role for a physician or a surgeon.

In addition to a hierarchy based on inheritance, Moffett proposes two further hierarchies. The first is an activity hierarchy; in this hierarchy, permissions are bundled together to form a collection of permissions that are needed to carry out various tasks that logically belong together from an organisational standpoint. For example, a sequence of tasks may be required to provide a specific customer service, such as booking a flight. The second is a supervision hierarchy; this hierarchy is what is normally considered to be the organisational hierarchy in that it represents the lines of supervision showing the seniority of the members of staff. This is the hierarchy that can also be used to differentiate permissions between junior and senior members of staff.

This actually says something significant about the meaning of roles. In Moffett's hierarchies there are roles based on the function of the individual, task, and seniority, each potentially part of a different hierarchy. In other words, different organisational characteristics/parameters are being captured in the form of roles, and the relationships between them.

Park *et al.* (2004) differentiate between organisational roles, system roles, and enterprise roles. Organisational roles represent positions in an organisation allowed to carry out the core activities; enterprise roles represent members of teams to perform a temporary task such as a project; and system roles represent supportive roles such as administrators. Each of these role types has its own hierarchy. These separate role hierarchies identified by Park *et al.* are similar to the classifications of different parts of the organisations as identified by Mintzberg (1978), who identifies the operational core and management as the main hierarchy, with separate authority structures for the

"technostructure"; i.e. specialists maintaining the organisation's operations infrastructure; and support functions.

Bacon *et al.* (2001) demonstrate how different types of roles based on function and seniority can be combined. In order to be assigned certain roles, a user must have been assigned other prerequisite roles, for example, a doctor can only be assigned the role of senior haematologist if the roles senior doctor and haematologist have already been assigned. Here the role senior haematologist is in effect a composition of the functional role haematologist, and the positional role senior doctor. Similarly a major European bank has defined positional and functional roles in its RBAC system (Schaad *et al.*, 2001), where positional roles represent positions of authority, and functional roles the membership of functional groupings. Abdallah and Kayhat (2004) propose parameterising roles, where parameters can be used to represent different aspects of the organisational context, such as a level in the organisational hierarchy, or a department.

Other work in this area has identified how roles can relate to context. Bertino *et al.* (2000) describe how temporal constraints can be defined for roles, for example, when a role is activated for a shift, and then subsequently deactivated. In addition, an administrator can activate roles *ad hoc*. Covington *et al.* (2001) have explored applications for the home, and suggest how environmental roles could be useful. Access can be permitted based on environmental factors, such as location or time of day. Georgiadis *et al.* (2001) combine contextual information with team based access control. Team based roles identified by Thomas (1997) are useful for collaborative working environments, where users are assigned to teams and get access to the team's resources. This can be combined with other contextual information, such as location or time intervals. Yao *et al.* (2001) present an access control model (OASIS), whereby

users can activate roles, provided they satisfy prerequisite conditions, such as having an appropriate qualification, assigned function, task competence, or environmental constraint.

Researchers have also explored techniques for deriving roles. Role Engineering, as outlined by Coyne (1996 ), is a systematic process of identifying the activities of a single user and defining this as a role. Fernandez and Hawkins (1997) propose deriving roles from use case actor definitions, and Neumann and Strembeck (2002) propose deriving roles from scenarios of the work-process. All these approaches focus on deriving roles from tasks, and largely ignore the wider organisational context.

## 3.3 Relating Actors to the Organisational Context in Requirements Models

In the last chapter we reviewed some key contributions with regard to modelling access policies in requirements models. Having now additionally reviewed the literature on organisations and security, it is now useful to revisit and explore in more depth the problem we highlighted in chapter 2 with regard to access policy definitions using actors.

The actor or agent is a key to the definition of access policies. Access policies are restrictions on the user, and in requirements models agents or actors are used to represent human users. Actors or agents are generally synonymous with roles. Only in scenarios does it make sense to model a specific individual, in abstract models we use roles. In a library system, an example used to demonstrate KAOS (Dardenne *et al.*, 1993), two actors defined are the librarian and the borrower. Herein however lies a significant problem, and that is how do we define a role precisely? In the library system it maybe obvious, but this is not always the case, as He *et al.* (2006) in their

approach to role engineering highlighted in applying their approach in practice. Certain assumptions maybe made about a role by a stakeholder that are not explicit in the name. Taking the example of the family doctor presented by Liu *et al.* that we reviewed in the previous chapter; maybe there are doctors of different seniority in the practice, and in describing what a family doctor can do, the stakeholder may implicitly be referring to a senior doctor, but unless this is raised by the analyst, this will be missed out. Terminology may also be important; for example is family doctor the only term we can use to describe this role? What about general practitioner, or just plain doctor? Would these be wrong or right?

The problem becomes clearer when we consult the literature on role theory (Biddle, 1979). A role is nothing more than expectation of some behaviour, and has as much to do with perception as a formal definition. What is this behaviour, and where do these expectations arise? The fact is that roles are social phenomena and as such do not have precise formal definitions. The problem is compounded by the fact that users will adopt multiple roles and vice-versa; furthermore individuals will interact with each other depending on the roles that they are adopting (Handy, 1985).

However languages specifically developed for specifying access policies, such as Ponder, can map the division of work in terms of subject and target domains, and the organisational hierarchy can be modelled as management structures. Thus in Ponder, for example, we can define policies that satisfy the principles of minimum privileges, as well as delegation and revocation of authority, whereas in requirements models, such as i* framework and KAOS, we can not. Researchers in security have also explored the problem of what roles are, Moffett and Lupu's separation of concerns reduce the ambiguities that can occur in RBAC policy definitions.

## 3.4 Chapter Summary

In this chapter we began with a review of the organisational literature. Organisations vary but we can generalise about organisational structure; in particular organisational groupings are based around functional and marketing characteristics. An important way in which organisations vary is through culture, ranging from informal structures to larger more formal structures where work is governed by procedures. These principles provide a reference point for identifying modelling concepts.

We reviewed access control frameworks, and in particular the concepts used in policy languages. Groups are a key way in which policies are formulated, and which map onto the groups of an organisation; however groups themselves are inadequate for defining requirements, as they are based on instances, and hence entail defining policies for every group in the organisation. Roles can provide a useful abstraction, and the research in security has demonstrated how they can be used to relate users to the organisational context, separating the concerns of authority and function.

We then revisited the problem of using actor definitions in current requirements models as a basis for defining access policies. We observed that an actor is in effect a role, but in contrast to policy frameworks, requirements modelling frameworks do not have an adequate link to the organisational context. A role is a social phenomenon open to interpretation, and requirements modelling approaches do not offer us a way of ensuring that the actor's role definition is precise. In policy frameworks such as Ponder, roles and subjects are directly derived from the organisational context, and thus it is possible to define policies that are based on management control principles such as the minimum privileges principle and delegation.

There is a need to define the organisational context, to which we need to link to actor definitions, and to identify concepts with which we can model this organisational context. It is the organisational context, which potentially provides a framework of reference to clearly identify the obligations assigned to a specific user, and the authority that he can exercise.

# Chapter 4

# The i* Framework and Formal Tropos

One of our research objectives is to extend an existing requirements modelling approach. In this chapter we review in more detail the i* framework, a requirements modelling approach that we extend in chapter 6. In chapter 2 we reviewed different approaches to modelling. We identified different paradigms; including goal based approaches, approaches around the organisational context, and scenarios. In particular we identified a class of requirements modelling approaches, which are based on the organisational context. Of these approaches we selected the i* framework for extension as it has advantages over other approaches; it is supported by a formal model and tools for representing and analysing models.

Originally devised as an early approach to requirements analysis to explore alternatives in the early phases of requirements analysis, as we mentioned in chapter 2, it has recently become incorporated into the Tropos methodology. The Tropos methodology covers early and late requirements analysis, as well as architectural and detailed design. However we are really only interested in the early requirements phase and the i* framework, as we are focusing on the specification of requirements of access policies. The i* framework is a semi-formal approach to requirements analysis based on the use of diagrams to model requirements; recently, however, the i* framework has been formalised in the Tropos methodology and is known as formal

Tropos. The formalisation of the i* framework is particularly interesting as we can reason about it.

In this chapter we first of all review the i* modelling language in more detail, and then the formal version of the i* framework, formal Tropos.

## 4.1 Introduction to a Case Study: A Software Project

To illustrate the i* framework and formal Tropos we use a case study. This case study is based on an example in the literature (Sandhu *et al.*, 1996) and the author's own experience. It concerns a software project, and access to the project resources, relating the roles of project manager, programmer, and test engineer.

## 4.2 The i* Framework

The i* framework, which we reviewed in chapters 2 enables us to model dependencies, which is done at two levels.

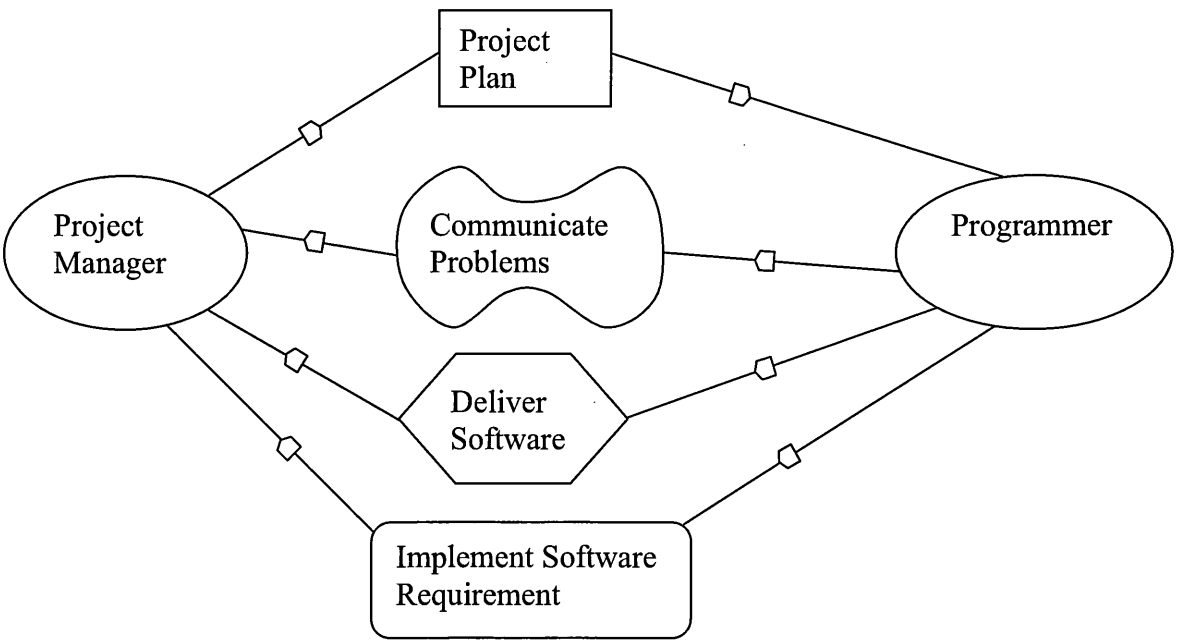



**Figure 4.1 i* Framework Strategic Dependency Diagram**

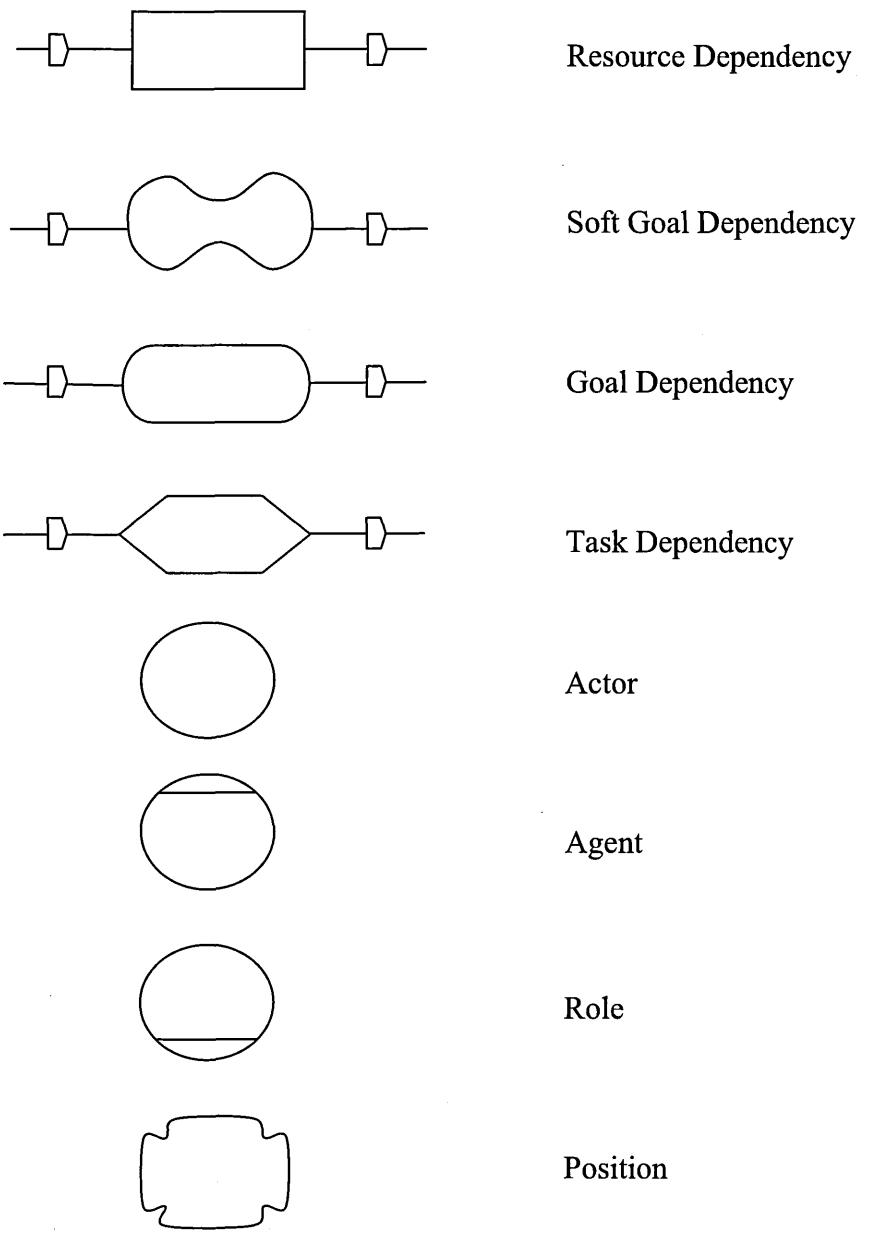



**Figure 4.2 i* Framework Symbols in Strategic Dependency Diagrams**

The first level is modelled as a Strategic Dependency (SD) diagram, as shown in figure 4.1. The meanings of the symbols are shown below in figure 4.2. The SD diagram models how actors depend on one another, to carry out tasks, provide resources or fulfil goals.

An actor can actually be an agent, role or position, where an *agent* is a physical entity such as a human, a *role* is defined as an abstract actor that can be adopted by a physical agent, such as conducting a task, and a *position* represents a set of roles that can be assigned to an agent. So for example we could define Jim Smith as an agent, who is as an instance of the agent Qualified Software Engineer representing a person with the capability to program, and who occupies the position of Programmer, and a couple of roles associated with the position of Programmer, such as Implement Requirement, and Write Code. This is represented in figure 4.3.

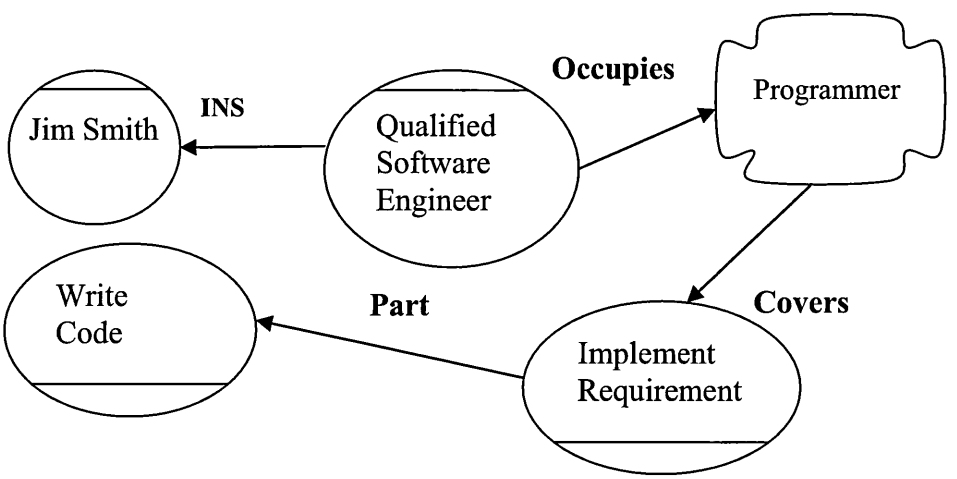
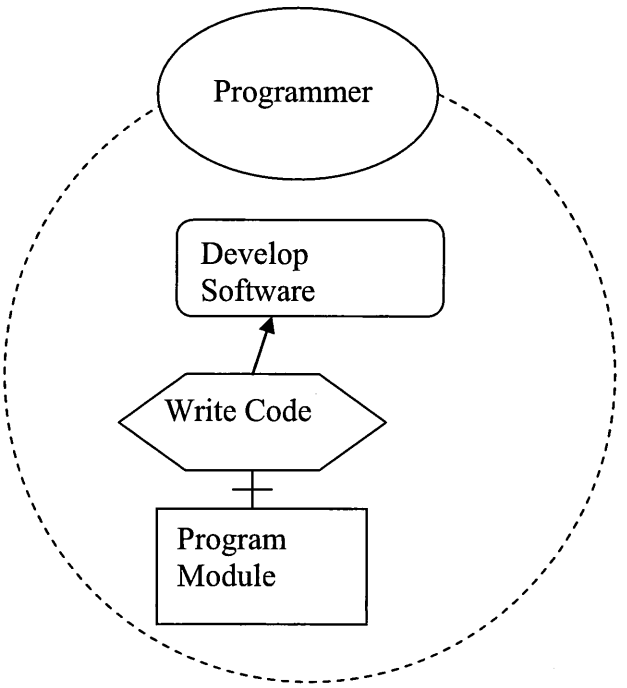



**Figure 4.3 i* Framework Actors**

At another level is a Strategic Rationale (SR) diagram, which explores a single actor, modelling the internal relationships within an actor, an example of which is shown in figure 4.4. This diagram shows an actor's boundary, and a goal internal to the actor Programmer to achieve the goal of providing Software, which is dependent on carrying out the task of Writing Code. This task depends on the resource Program Module. In chapter 2 we described how the actor boundary as proposed by Liu *et al.* (2003) can be used to define a policy, where the actor has the right to perform tasks and access resources defined within the boundary.




The symbols have the following meaning:

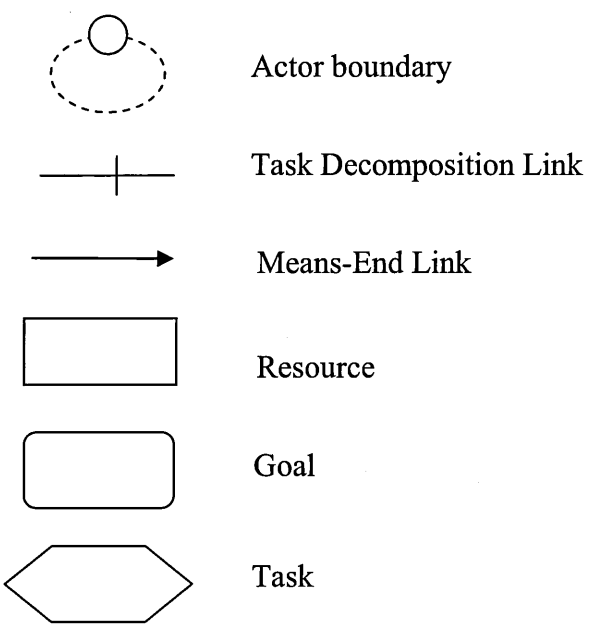



**Figure 4.4 i* Framework Strategic Rationale Diagram**

In representing an access policy this diagram exhibits a similar weakness to the diagram we reviewed in chapter 2 of the access policy for a family doctor. This policy describes the access to the resource of a Program Module, to perform the task Write

Code, but this restriction is only complete when it includes the organisational context of Project, which is not represented on this diagram. A programmer should not be able to access program modules of any project.

## 4.3 Formal Tropos

The use of formal Tropos in order to extend the i* modelling language has been proposed by Fuxman *et al.* (2001). The key motivation for the extensions to the i* framework is to utilise the advantage of formal methods in order to perform reasoning on requirements models, and hence verify the properties of a model and identify inconsistencies.

Formal Tropos describes the relevant objects of the modelled domain and has two layers. The outer layer models the objects as classes, whereby a class can be an actor, dependency or entity. Entities do not exist in the i* framework and are used to represent elements that do not appear in the model as they are not directly related to the actors' strategic goals. Attributes in the class definitions represent relationships between classes. The intentional relationships between actors are represented as dependencies, which can be goals, soft-goals, tasks or resources. The inner layer of the formal Tropos language is identical to the inner layer used in the KAOS framework, which is a first order predicate language with temporal constraints. The following standard first order predicate and logical operators are used:

$\forall$ for all, $\exists$ there exists, $\Rightarrow$ implies, $\wedge$ and, $\vee$ or, $\Leftrightarrow$ equivalent, $\neg$ not

Examples of the temporal operators are as follows:

| | |
|---|---|
| $\Diamond$ at some time in the future | $\blacklozenge$ at some point in the past |
| $\circ$ in the next state | $\bullet$ in the previous state |
| $\Box$ always in the future | $\blacksquare$ always in the past |

$\Diamond_{\leq d}$ sometime in the future within deadline d

$\Box_{\leq d}$ always in the future up to deadline d

The inner layer of Tropos is used to define constraints on the class definitions. First order logic quantifiers $\forall$ and $\exists$ can be used to range over all instances of a class; in addition each instance of a class may express properties about itself using the operator self.

In addition to class definitions, Tropos provides formulae that describe properties of the system as a whole. There are system invariants that hold in all states of the system, there are system assertions that hold on all executions, and there are system possibilities that hold on some possible behaviours of the system.

To demonstrate formal Tropos we have translated a part of the i* dependency and strategic rational diagrams above as follows:

**Entity** Software Requirement
  **Attribute** approved: Boolean
  **Attribute** submitted: Boolean
  **Attribute constant** pm: Program Module
**Entity** Program Module
  **Attribute** completed: Boolean

**Actor** Programmer
  **Goal** Develop Software
    **Mode** Achieve
    **Fulfilment definition**:
      $\forall$isr: Implement Software Requirement,
      isr.depender = self $\Rightarrow \Diamond$ isr.sr.pm.completed

**Actor** Project Manager
  **Goal** Manage Project
    **Mode** Achieve
    **Fulfilment definition**:
      $\forall$isr: Implement Software Requirement,
      isr.dependee = self $\Rightarrow \Diamond$ isr.sr.pm.completed

**Dependency** Implement Software Requirement
**Type** Goal
**Mode** Achieve
**Depender** Programmer
**Dependee** Project Manager
**Attribute** sr: Software Requirement
**Creation**
**trigger** sr.submitted
**condition** sr.approved
**Fulfilment**
**condition for depender** sr.pm.completed

In order to define the formal specification we have defined Software Requirement and Program Module entities. The entity Software Requirement has boolean attributes, approved and submitted to represent the states. The third attribute pm relates a Software Requirement to a Program Module. The keyword **constant** means that an association between a Software Requirement and a Program Module can not be changed.

We have defined the goal dependency Implement Software Requirement between the dependee, Project Manager, and the depender, Programmer. This has a modality of achieve. That means this goal needs to be satisfied once. The other modalities are avoid, which is a goal to prevent a specified condition from occurring, maintain is a goal that continuously has to be satisfied, and maintain and achieve, which is a goal that has fulfilment conditions that continuously need to be satisfied.

The goal dependency in the example above has a creation condition, which is a constraint imposed when a dependency is created. In this example the software requirement must have been approved. There is also a creation trigger, which represents the condition that will cause the dependency to be generated. In this case this occurs when a sofware requirement is submitted. Fulfilment definitions are constraints that define conditions for the satisfaction of a dependency. In this example

this is achieved when the depender, the programmer, has completed the the program module associated with the implementation of the software requirement.

In the example we have also defined a goal, Manage Project, associated with the project manager. This goal has a fulfilment condition that states, for all dependencies of type Implement Software Requirement for which the project manager is the dependee, the program module associated with the software requirement must be completed at some time in the future. A second goal, Develop Software, has been associated to the programmer; this is almost identical to the previous goal defined for the project manager, except that the programmer is the dependor.

It should be noted that there are number of features in the i* framework which have not been incorporated into formal Tropos. The differentiation between the different types of actors, (position, role and agent), have not been included. Formal Tropos does not currently support the modelling of SR diagrams; i.e. means-end relationships and task decomposition links that exist between tasks and the other elements of the framework. This means that currently access policies, as proposed by Liu *et al.* (2003), can not be defined using formal Tropos.

## 4.4 Formal Analysis in Tropos

A tool has been developed for analysing specifications in Tropos. The tool can analyse all possible executions of the system. It checks that assertions and possibility formulae are enforced, it does this by looking for counter examples. If a counter example is found it is reported.

For example in the model above we can define an assertion, that a progam module should not be completed if the software requirement has not been approved, which would be specified as follows:

**Dependency** Implement Software Requirement
**Fulfilment**
**assertion condition for dependee**
sr.pm.completed ⇒ sr.approved

This assertion is satisfied due to the constraint in the creation condition.

## 4.5 Chapter Summary

In this chapter we have reviewed the i* framework and formal Tropos, highlighting the weakness of defining access policies in an i* model through a case study. We first gave an overview of the two types of dependency diagrams that exist in the i* framework: SD diagrams and SR diagrams. We then reviewed formal Tropos, a formal specification language to represent i* models. We highlighted the fact that formal Tropos does not currently allow us to represent SR models. We then reviewed how analysis in formal Tropos is supported by a tool.

# Chapter 5

# A Framework for Modelling Access Policies

This chapter describes our framework for modelling access policies. This framework consists of a meta-model that describes domain independent abstractions, and how they relate to one another. These abstractions allow us to represent the organisational context, and to model how users relate to the organisation. It includes a set of constructs for modelling policies and defining scenarios. In particular, we present policy constructs that enable us to define restrictions that satisfy the principle of minimum privileges. Invariants allow us to check the correctness of domain definitions, and to check that scenarios are consistent with policy definitions.

We first introduce a case study. In section 5.2 we justify and introduce the Z notation we have used to develop the framework. In section 5.3 we then describe the rationale of the framework; i.e. the principles for defining the organisational context with an explanation of the organisational domain abstractions, and how we relate them to users and define policies. In section 5.4 we present the meta-model, comprising meta-concepts and relations between the meta-concepts, and the constructs that enable us to define policies and scenarios. In section 5.5 we present the heuristics for defining and verifying policies. We present these heuristics as a set of steps that could be used as a basis for a method, using the case study as an example. We conclude with a chapter summary.

## 5.1 Introduction to a Case Study in the Medical Domain

The presentation of the framework in this chapter is illustrated through a case study in the medical domain, concerning the minimum privileges policy for access to medical and nursing records. It is based on discussions with medical and nursing staff who have worked in hospitals. The hospital and members of staff that we use in this example are hypothetical.

## 5.2 Framework Modelling Notation

### 5.2.1 Review of Formal Notations

In developing the meta-model for the framework, a language is required to express and reason about it. An important selection criterion is the degree of formality of the language. Requirements can be expressed using formal specification languages, semi-formal approaches, or informal representations (Jarke *et al.*, 1993). The purpose of our framework is to define what meta-concepts we require to represent policies, and how the constructs can be formulated. Formal notations provide us with the key advantage of precise and unambiguous representation. Another objective of our research is to reason about our models. In particular we want to be able to verify the consistency of scenarios with policies. Formal notation enables us to do this more easily.

There are however many different formal notations to choose from (Clarke & Wing, 1996; van Lamsweerde, 2000a). In this research we are primarily concerned with defining restrictions to maintain a secure state. There are a number of languages that are based on the paradigm of defining admissible states; these include Z, VDM and B (van Lamsweerde, 2000a), and their variants (Büchi, 1998). They enable us to

define invariants constraining the system at any point in time. The languages are very similar; for example, B was derived from Z (Carnot *et al.*, 1992). Z however does differ from VDM and B in that invariants can be defined that apply to an entire schema, which means that preconditions in Z are implicit. In contrast, in VDM (Jones, 1990) and B (Carnot *et al.*, 1992), the preconditions need to be explicitly defined for functions. This is one of several aspects why VDM and B are more akin to conventional programming languages, and hence why refinement into code is easier with these languages. Indeed one of the motivations for B was improved refinement into code, and hence programming constructs were added to the mathematical notation. We, however, are interested in defining the properties of a framework, and the nature of Z as a pure specification language (Büchi, 1998), with a purer mathematical notation, is therefore more appropriate.

### 5.2.2 Z Notation

We only use a subset of the Z languauge, which we now introduce. A comprehensive description of the Z notation is given by Spivey (1992).

| | |
|---|---|
| [A] | basic type definition; A is a basic type. |
| a: A | variable definition; this introduces the variable a of type A. |
| $C \cong [ a: A; b: B ]$ | definition of a composite type; C is composed of two components, a and b of types A and B respectively. |
| c.a | selection operator for a composite type; given a variable c of type C, c.a selects the component a. |
| $A \wedge B$ | conjunction operator; this expression is true if both A and B are true. |
| $A \vee B$ | disjunction operator; this expression is true if either A or B is true. |
| $A \Rightarrow B$ | implies; if A is true then B is also true. |

| | |
|---|---|
| $A = B$ | equality; A equals B. |
| $A \neq B$ | negation of equality; A is not equal to B. |
| $\forall a: A \cdot B$ | universal quantifier; for all a in A, B is true. |
| $\exists a: A \cdot B$ | existential quantifier; for some a in A, B is true. |
| $\nexists a: A \cdot B$ | negation of the existential quantifier; for no a in A, is B true. |
| {a, b, c} | set declaration; this declares a set containing the members, a, b, and c. |
| $a \in A$ | set membership; a is a member of A. |
| $A \subseteq B$ | subset; A is a subset of B or equivalent. |
| $\varnothing$ | empty set symbol; this represents a set that contains no elements. |
| $A \leftrightarrow B$ | binary relation; this represents a set of relations between a set of type A, the domain, and a set of type B, the range. |
| $A \rightarrow B$ | function; this represents a set of relations between a set of type A, and a set of type B. Each member of A relates to one member in B. |
| $A \nrightarrow B$ | partial function; this represents a set of relations between a set of type A and a set of type B. Each member of A relates at most to one member in B; i.e there may be members of A that do not relate to members of B. |
| $a \mapsto b$ | maplet; this represents an element in a relation, denoting that a relates to b. |
| **dom** R | domain of a relation; this denotes the set of all the elements in the domain of a relation. If the set R relates X to Y, the domain is X. |
| **ran** R | range of a relation; this denotes the set of all the elements in the range of a relation. If the set R relates X to Y, the range is Y. |
| $R^+$ | transitive closure of a relation; if R is a relation that relates elements of the same type, and contains $x \mapsto y$ and $y \mapsto z$, then $R^+$ contains R and all indirect relations; i.e. $x \mapsto z$. |

| | |
|---|---|
| $R^*$ | reflexive transitive closure of a relation; if R is a relation that relates elements of the same type, then $R^*$ contains $R^+$ and in addition relates all members of X to themselves i.e.; $x \mapsto x$. |
| R ( A ) | relational image; this represents a subset of the range of the relation R. This subset contains those members of the range of R, which are mapped to members of the domain in R contained in the set A; i.e. A is a restriction on the domain of R. |

## 5.3 Rationale of the Framework

Before formally defining our framework, we first explain its rationale; i.e. what is it that the framework is to achieve, and the reason for the choice of meta-concepts. Our framework has been developed to focus on policies that ensure compliance with the principle of minimum privileges. The selection of the minimum privileges principle as a focus of the research was justified in chapter 2.

The meta-model of our framework allows us to define organisational policies, whereby a policy in this meta-model is an assignment of a task to a role. This assignment of a task to a role reflects the assignment of an obligation to a member of the organisation by its management to carry out an activity. As we reviewed in chapter 3, large organisations have formal structures, activities are assigned to roles rather than specific individuals. Which ever individual adopts the role will then assume the obligation to execute duties associated with that role, and therefore will need access to resources associated with those duties. The principle of minimum privileges states that individuals must only have access to resources they require to execute their duties. Hence a set of policies satisfying this principle will define precisely which tasks on which assets will be executed by which roles.

Whereas tasks and assets are unambiguous, and as we reviewed in chapter 2, their definition is subject to analysis techniques that are well established, the definition of

roles is problematic due to the fact that roles are subjective social phenomena. In order to solve this problem the framework relates roles to the organisational context.

In chapter 3 we identified two key dimensions by which organisations are structured, along the lines of authority, and according to the division of work, and the fact that the division of work is based on a mixture of functional and market characteristics. The organisation is composed of groups each assigned a function and market with a hierarchical structure. The key therefore to defining a role precisely is to link a role definition to the function, an organisational domain for the market, and the level of authority. We can illustrate this with an example of a hospital.

Within a hospital two key functions are medicine, the function performed by doctors, and nursing, performed by nurses. These represent two key functional groups within the hospital. Nurses are assigned to wards, which effectively represent a division of work based on markets, where the patients in a ward represent the market being served by the nurses, and would be represented as an organisational domain in our framework. Doctors are formed into groups led by consultants, and each consultant carries responsibility for patients referred to him. Thus patients referred to the consultant represent the market served by the consultant, and his subordinates within the organisational domain of the consultants group. Within these groups are seniority levels, such as the consultant, who manages registrars, and within the nursing function a ward sister is in charge of staff nurses on the ward. So in this example we can define the role of consultant medical specialist, which has a function of medicine, an organisational domain of consultant group, and authority level of consultant, and a further role of senior ward nurse that has a function of nurse, an organisational domain of ward, and authority level of sister.

Using these roles we can then define policies, as we described above, by assigning them to tasks. So we can define the task, read medical record, which has an asset dependency of medical record, and the policy is then defined as an assignment of the role consultant medical specialist to the task, read medical record. The policy allows a consultant to read medical records of patients referred to him; i.e. those medical records of his referred patients are assigned to the same organisational domain.

## 5.4 Framework Meta-Model

### 5.4.1 Metal-Model Overview

The framework consists of a meta-model that describes domain independent abstractions, which we refer to as meta-concepts, and how they relate to one another. Policies are defined using domain concepts by instantiating the meta-concepts. Examples of meta-concepts include role, and organisational function. An example of a domain concept is a consultant medical specialist, which is an instantiation of the meta-concept role. A policy is verified by instantiating domain concepts, and checking whether the policy is consistent with that instantiation. For example, Greenfield Hospital is an instantiation of the organisational domain, hospital.

### 5.4.2 Metal-Concepts

We now introduce the formal definitions of the meta-concepts. Since policies define restrictions on access to valuable information assets, and such access is required to carry out tasks, we need the meta-concepts of asset and task:

[Asset] An asset represents a resource that we wish to protect.

[Task] A task represents the activity that an organisational unit or individual carries out.

In order to describe restrictions with respect to individuals, we also need the meta-concepts of agent and role:

[Agent] An agent represents a physical person.

[Role] A role represents an assignment of an obligation, of performing some function, which is a composite element representing the organisational function, organisational domain, and authority.

As we need to link a role to the macro-organisational context, we also need some additional meta-concepts:

[Org_Function] An organisational function represents a functional grouping within an organisation. Members of a functional grouping will be expected to carry out tasks that will be assigned to this group.

[Org_Domain] An organisational domain represents a market based grouping; i.e. a grouping that is assigned a market to serve, such as a set of clients in a specific geographic location. An example of this would be a hospital, which serves patients in its locality.

[Authority] A level of authority represents the seniority of a role.

The meta-concept role as described above is a composite of authority, organisational function, and organisational domain, and is defined formally as follows:

Role ≙ [ authority: Authority; org_function: Org_Function; org_domain: Org_Domain ]

A key decision in defining this meta-model was to define a role as a composite element representing the organisational function, organisational domain, and authority. As we reviewed in chapter 3, a role can have many meanings, it is indeed anything that conveys behaviour, which can lead to ambiguous definitions. Defining a role as a composition of organisational elements removes this ambiguity. It provides a

precise link to the organisational context. These organisational elements are defined, so that they can be derived from the organisational structure.

Organisational functions are derived from groupings based on function; organisational domains are derived from groupings based on marketing characteristics, and authority based on the hierarchical relationships within these groups. Referring to the example introduced earlier, in a hospital, the two key organisational functions are nursing and medicine. The organisational domain represents the grouping based on market characteristics, in this case the hospital that serves a local community, which itself is part of a regional health authority. The hospital is divided into wards, which are organisational domains to which nurses are assigned.

### 5.4.3 Inheritance and Aggregation Hierarchies

We have also introduced inheritance hierarchies for organisational functions, and aggregation hierarchies for organisational domains and tasks. The inheritance hierarchy for organisational functions was introduced to model the characteristic that groups are often structured according to an increasing level of specialisation. Thus the medical specialists of a hospital are divided into physicians and surgeons, which are in turn divided into groups with a further level of specialisation, such as cardiologists or haematologists. The inheritance property allows us to model common characteristics of medical specialists. This is useful when we define policies. For example all medical specialists keep medical records, so we can define a policy for medical staff, without the necessity to define a policy that is repeated for each speciality. Since policies are defined on roles, the inheritance relationship also exists between roles, where if a role inherits from another a role it has the same

organisational domain, and authority, but different organisational functions that are themselves related through inheritance.

The inheritance between organisational functions is formally defined as follows:

$inhf: Org\_Function \rightarrow\!\!\!\!\!\!+\ Org\_Function$

The inheritance of roles is represented by the following function:

$inhr: Role \rightarrow\!\!\!\!\!\!+\ Role$

The domain role inherits from the range role, and likewise for organisational functions.

Formally, to determine whether a role inherits from another, we need a relational image on a transitive closure. So, to specify the condition that a role, role2, is inherited by role1 we can write:

$role2 \in inhr^{+}(\{role1\})$

If a role has an organisational function that is inherited from an organisational function of another role, and these two roles have an identical organisational domain and level of authority, then there exists likewise an inheritance relationship between the two roles.

The aggregation hierarchy for organisational domains allows us to model an organisational structure based on marketing characteristics. This enables us to capture the division of work based on markets (Mintzberg, 1992). As we described above, hospitals have wards each representing an organisational domain, and the hospital is itself an organisational domain. Thus if we define a policy for a hospital manager to access staff records, then this allows him to access records within each ward. This aggregation hierarchy allows us to define this property. Formally this is expressed as:

$aggd: Org\_domain \rightarrow\!\!\!\!\!\!+\ Org\_Domain$

This function has the aggregated organisational domain as the range.

There are a number of constraints related to these relationships that are useful, to ensure consistency. Organisational functions and roles can not inherit themselves, similarly an organisational domain can not be an aggregation of itself. These three constraints are defined respectively as follows:

$\forall$of: org_function • of $\notin$ inhf$^+$ ( {of} ) defines an organisational function can not inherit itself.

$\forall$r: role • r $\notin$ inhr$^+$ ( {r} ) defines a role can not inherit itself.

$\forall$od: org_domain • od $\notin$ aggd$^+$ ( {od} ) defines an organisational domain can not be an aggregation of itself.

The following invariant states that a role, role2, that is inherited by role1 must have an organisational function that is inherited by the organisational function assigned to role1, and must have an identical level of authority and organisational domain.

$\forall$role1; role2: role • role2 $\in$ inhr$^+$ ( {role1} ) $\Rightarrow$ role2.org_function $\in$
inhf$^+$ ( {role1.org_function} ) $\wedge$ role2.org_domain = role1.org_domain $\wedge$
role2.authority = role1.authority

### 5.4.4 Levels of Authority

The meta-concept Authority, as part of a role, represents the seniority of that role. If we want to represent the organisational hierarchy as proposed by Moffett & Lupu, (1999), then we need to identify the hierarchical relationships between roles that represent the lines of authority. In fact for the purposes of modelling minimum privileges we do not need to represent the hierarchy; however, we do need to represent it if we are to extend our framework to model other principles, such as

delegation. So, in order to capture the lines of authority, we introduce the function senior, which models the seniority as follows:

$\text{senior: Authority} \rightarrow\!\!\!\!\!\!+\ \text{Authority}$

The function senior maps junior levels of authority to senior levels; i.e. a junior level can at most map to one senior level in a single organisational domain. In matrix or project based organisations an individual can be assigned to more than one group (Handy, 1985), and hence report to more than one superior. This can potentially be represented in this framework by assigning an agent multiple roles in different organisational domains.

There is one constraint that we have defined with respect to seniority relationships, and that is that a level of authority can not be senior to itself. This is defined as follows:

$\forall a: \text{Authority} \cdot a \notin \text{senior}^{+} (\{a\})$

### 5.4.5 Organisational Assets and Tasks

Tasks can often be subdivided. It is important to model this, because if a task is assigned to an individual, then this will entail carrying out all its constituent subtasks. This subdivision can be represented as an aggregation hierarchy for tasks, which enables us to define a policy for a composite task. Formally the task aggregation is defined as follows:

$\text{aggt: Task} \leftrightarrow \text{Task}$

Sub-tasks are defined as the range.

Tasks can be divided down to the lowest level of granularity, to the point at which they represent a single action, where the action can be assigned to an asset or group of assets. Tasks at the lowest level in the aggregation hierarchy can be mapped onto

actions or tasks in existing requirements models. The relationship between tasks and assets is represented by a task asset dependency relationship:

task_asset_dependency: Task $\leftrightarrow$ Asset

Assets belong to organisational domains. This reflects the subdivision of work based on marketing characteristics. We represent this as:

asset_domain: Asset $\rightarrow$ Org_Domain

There is a constraint that we have defined with respect to task aggregation and that is that a task can not be aggregated by itself:

$\forall t$: Task $\cdot$ t $\notin$ aggt$^{+}$ ( {t} )

### 5.4.6 Policy Definitions

We define policies using the following composite type:

Authorisation_Policy $\widehat{=}$ [ role: Role; task: Task ]

Within this policy, there are two implicit assumptions: firstly, the policy applies to any subtasks of the task in the policy; and secondly, the organisational domain in the role of the policy applies to all assets associated with the task through the relation:

task_asset_dependency: Task $\leftrightarrow$ Asset

### 5.4.7 Policy Verification

We now explain how our meta-model can be used to verify that an instantiation is consistent with a policy specification.

First, we create an instantiation, which in effect is a simple scenario of an agent executing a task. In creating this scenario, not all domain concepts can be instantiated. The level of authority and the organisational function are constants. For example, if we define an organisational function of nursing, then this organisational function will

not change for the instantiation. Instantiations are required of organisational domains, roles, tasks, assets, and agents. We decided to use instantiation relationships between elements of the same type, which is an approach adopted in current requirements modelling frameworks such as the i* framework and KAOS. We initially explored using separate types for instantiated elements, and found that this added unnecessary complexity; in any case we need instantiation relationships, and this alone allows us to differentiate between domain definitions and instantiated elements.

An organisational domain instantiation will represent a specific organisational unit. For example, a hospital has wards, a ward is a domain description of an organisational unit, but ward A is an instantiation. Since an organisational domain is a composite part of a role, roles also need to be instantiated. So, if we define a nurse as an abstract role, then an instantiation of this would be a nurse in ward A.

The instantiation of organisational domains is defined formally as follows:

insd: Org_Domain ⇸ Org_Domain

Where the domain Org_Domain is instantiated from the range.

The instantiation for roles, tasks and assets are defined likewise, respectively as follows:

insr: Role ⇸ Role

inst: Task ⇸ Task

insa: Asset ⇸ Asset

To ensure consistency we have defined the following constraints on these three relationships, which define that an instantiated organisational domain, role, task or asset can not be instantiated from an organisational domain, role, task, or asset respectively, that itself is an instantiation.

$\forall od1; od2: Org\_Domain \cdot od2 \in insd(\{od1\}) \Rightarrow insd(\{od2\}) = \varnothing$

$\forall r1; r2: Role \cdot r2 \in insr(\{r1\}) \Rightarrow insr(\{r2\}) = \varnothing$

$\forall t1; t2: Task \cdot t2 \in inst(\{t1\}) \Rightarrow inst(\{t2\}) = \varnothing$

$\forall a1; a2: Asset \cdot a2 \in insa(\{a1\}) \Rightarrow insa(\{a2\}) = \varnothing$

A further constraint is that an abstract role not be associated with an instantiated organisational domain, and also that an instantiated role not be associated with an abstract organisational domain. We recognise an abstract role if the role is not mapped to any other role through the relation insr; i.e. it is not instantiated from any role. Similarly we can identify an abstract organisational domain if it is not mapped to any other organisational domain through the relation insd. This constraint is defined as follows:

$\nexists role: Role \cdot (insr(\{role\}) = \varnothing \wedge insd(\{role.org\_domain\}) \neq \varnothing) \vee (insr(\{role\}) \neq \varnothing \wedge insd(\{role.org\_domain\}) = \varnothing)$

The following constraint is to ensure that organisational domain aggregation relationships do not relate instantiated organisational domains with abstract organisational domains. This constraint is defined as follows:

$\forall od1; od2: Org\_Domain \cdot od1 \in aggd(\{od2\}) \Rightarrow (insd(\{od1\}) \neq \varnothing \wedge insd(\{od2\}) \neq \varnothing) \vee (insd(\{od1\}) = \varnothing \wedge insd(\{od2\}) = \varnothing)$

To prevent instantiated roles being used in policy definitions we have the following constraint:

$\nexists policy: Authorisation\_Policy \cdot insr(\{policy.role\}) \neq \varnothing$

This basically states there is no policy for which the condition is satisfied that the role defined for the policy has been instantiated.

Instantiated roles are assigned to agents as follows:

$role\_assignment: Agent \leftrightarrow Role$

We also need to model the carrying out of a task by an agent. This we represent via a relation performs, which defines an agent performing a task:

performs: Agent ↔ Task

The task in this relation must be instantiated. This is given by the following constraint:

∀p: performs • ∀task: **ran** performs • inst ( {task} ) ≠ ∅

The assets to which agent has access are given in the task asset dependency, and must be instantiated; furthermore they must be instantiated from assets defined in the task asset dependency of the corresponding task from which the task was instantiated. This is given by the following constraint:

∀p: performs • ∀task: **ran** performs •
∀ins_asset: task_asset_dependency ( {task} ) •
∃asset: task_asset_dependency ( inst ( {task} ) ) • asset ∈ insa ( {ins_asset} )

The definitions above allow us to verify that a specific instantiation is consistent with a policy, through an invariant:

∀user: Agent; user_task: Task • user_task ∈ perfoms ( {user} ) ⇒
∃role: role_assignment ( {user} ) •
∃policy: Authorisation_Policy • policy.role ∈ inhr* ( insr ( {role} ) ) ∧
inst ( {user_task} ) ⊆ aggt* ( {policy.task} ) ∧
∀asset: task_asset_dependency ( {user_task} ) •
role.org_domain ∈ asset_domain ( {asset} )

This invariant is defined in the form: P ⇒ Q. P is the assertion that an agent has executed a task (though P can be a set of mappings between agents and tasks), and Q is the logical condition that there is a policy (or set of policies) that permits P. In order for Q to be satisfied a policy must exist for which three conditions must be satisfied. First, there is some role assigned to the user that is compatible with a policy. The user role is an instantiation of an abstract role, and if this role is equivalent to or inherited from a role defined in a policy, then the role is compatible with the policy. Second,

the task in the performs relation is instantiated from the task in the policy or one of its sub-tasks. Third, the assets being accessed through the task_asset_dependency must be in the same organisational domain as the user.

The invariant is therefore a check on the performs relation, which contains all mappings between agents in the system and instantiated tasks; i.e. tasks they can execute. If we define a mapping between an agent and a task in the performs relation, the invariant tells us whether it is permissible. If the invariant is true, then the task could be performed by that agent.

## 5.5 Heuristics for Defining and Verifying Policies

We now present some heuristics, illustrating how the framework can be used for defining and verifying policies, elaborating the example that we introduced with regard to policies for medical records earlier in this chapter. We present these heuristics as a set of steps that can form the basis of a method. The steps are as follows:

- Identify Organisational Groups
- Define Roles
- Identify Tasks and Assets
- Define Policies
- Verify Policies

In this section we now describe these steps in more detail.

### 5.5.1 Identifying Organisational Groups

The first step is to identify the organisational groups and how they relate to one another, in order to derive organisational domain and functional grouping definitions. The diagram in figure 5.1 shows the organisational structure of a hospital.

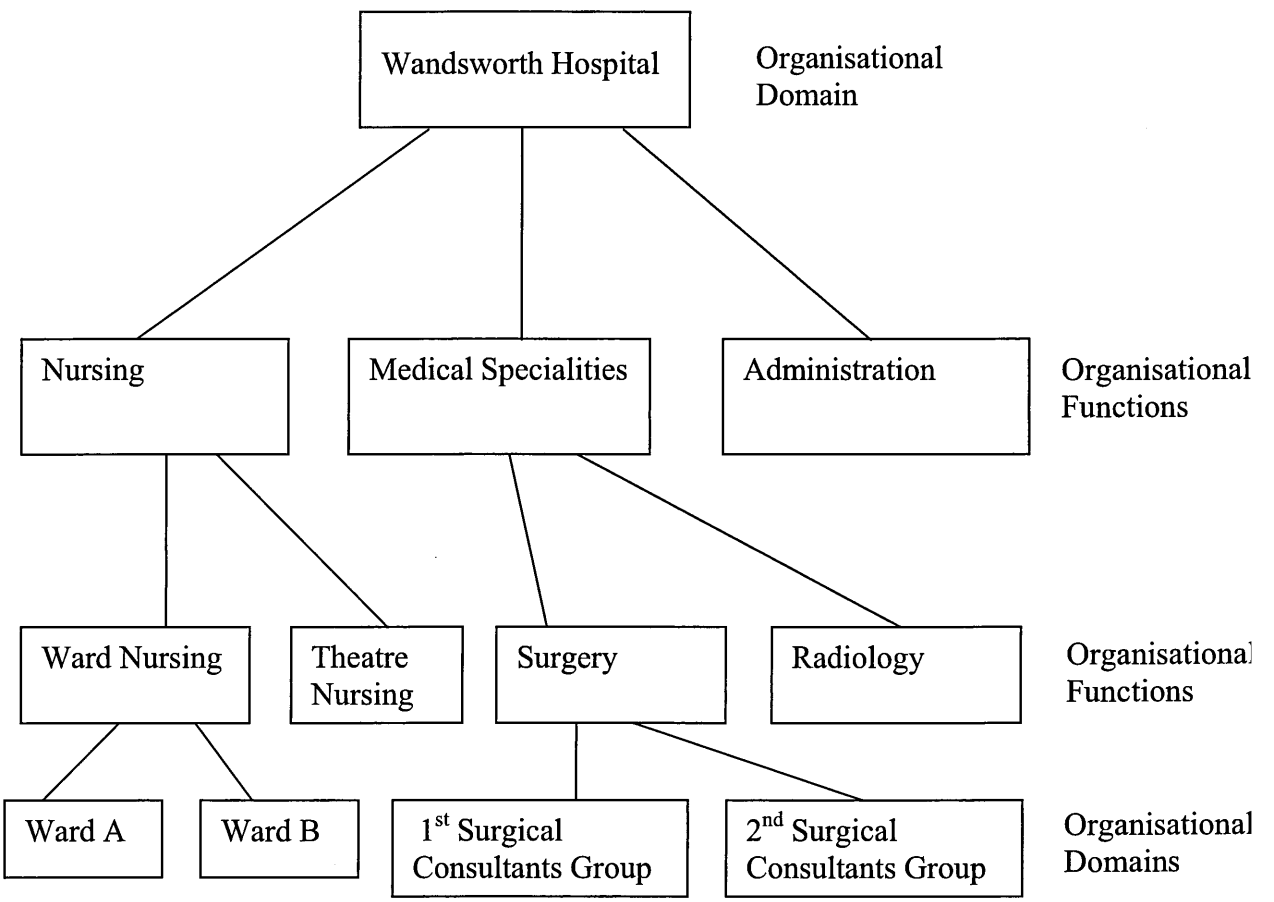



**Figure 5.1 Organisational Structure of a Hospital**

The hospital serves a local community and is effectively a grouping based on market characteristics, with each hospital having an identical functional structure. We therefore define hospital as an organisational domain, having the organisational functions of nursing, medical specialities, and administration. These functions are further specialised, so for example the nursing function is specialised into ward nursing and theatre nursing, and the function medical specialities into specialities

such as radiology and surgery. The ward nursing function is organised into wards, serving the patients stationed there, hence ward is an organisational domain; each medical speciality is divided into groups headed by a consultant serving sets of patients.

We can therefore derive the following organisational definitions. First of all we define the following organisational functions:

nursing, medical_specialities, ward_nursing, theatre_nursing, surgery, radiology: Org_Function

We can translate these into a specialisation hierarchy using the principle of inheritance as follows:

{ ward_nursing ↦ nursing, theatre_nursing ↦ nursing,
surgery ↦ medical_specialities, radiology ↦ medical_specialities } ∈ inhf

We can also identify the following organisational domains:

hospital, ward, consulting_group: Org_Domain

These relate to one another in an aggregation hierarchy as follows:

{ ward ↦ hospital, consulting_group ↦ hospital } ∈ aggd

### 5.5.2 Identifying Levels of Authority

Within each grouping there is a hierarchical structure, which will determine how the delegation of activities is carried out. Each consultant carries responsibility in the form of accountability for patients in his care and can delegate treatment to registrars within the group.

Levels of authority need to be assigned to groups. In the hospital we have the following authority levels for doctors and nurses:

consultant, registrar, staff_nurse, sister: Authority

Amongst medical practitioners a registrar is junior to a consultant, and in the wards a sister is senior to a staff nurse:

{ registrar ↦ consultant, staff_nurse ↦ sister } ∈ senior

### 5.5.3 Defining Roles

Having defined the above organisational context, we are now in a position to define roles, which are composite definitions. For example in the medical specialities we can identify two roles:

consultant_medical_specialist: Role
consultant_medical_specialist.authority = consultant
consultant_medical_specialist.org_function = medical_specialities
consultant_medical_specialist.org_domain = consultant group

registrar_medical_specialist: Role
registrar_medical_specialist.authority = registrar
registrar_medical_specialist.org_function = medical_specialities
registrar_medical_specialist.org_domain = consultant group

We note here that we have defined the role with the function of medical_specialities rather than surgery or radiology, since the organisational function medical_specialities is a generalisation of the other two functions, then the role is in itself a generalisation of surgery or radiology roles. This is a convenience if we want to use this role to specify the restriction of the access to medical records as the type of speciality is irrelevant. We also have included the rather abstract domain definition of consultant_group. Again this is convenience, rather than defining a role for each consultants group in the hospital, we just simply define it using the meta-domain of consultant_group. The relationship between a meta-domain and domain is through the instantiation relation.

Similarly for nursing we can identify the following two roles:

staff_nurse_ward, sister_ward: Role

For the role staff_nurse_ward the definition is as follows:

staff_nurse_ward.authority = staff_nurse
staff_nurse_ward.org_function = ward_nursing
staff_nurse_ward.org_domain = ward

### 5.5.4 Identifying Tasks and Assets

We now need to identify the tasks and their associated assets in the organisation. For example we can identify the tasks:

nurse_patient, treat_patient: Task

The tasks nurse_patient and treat_patient entail accessing nursing records and medical records respectively through the following sub-tasks, defined as follows:

read_nursing_record, create_nursing_record, read_medical_record,

create_medical_record: Task

The aggregation of these tasks can be modelled as follows:

{ nurse_patient ↦ read_nursing_record, nurse_patient ↦ create_nursing_record , treat_patient ↦ read_medical_record, treat_patient ↦ create_medical_record } ∈ aggt

For the tasks read_nursing_record and create_nursing_record we can identify the asset, nursing_record to which this task requires access, and likewise the tasks read_medical_record and create_medical_record require access to the asset medical_record. We then define these assets and task dependencies:

nursing_record, medical_record: Asset

{ read_nursing_record ↦ nursing_record, create_nursing_record ↦ nursing_record, read_medical_record ↦ medical_record, create_medical_record ↦ medical_record } ∈ task_asset_dependency

### 5.5.5 Defining Policies

Having defined the organisational context, roles and tasks, we now define policies. For example, in order to restrict the access of nursing records on the hospital wards we can define the following authorisation policy:

staff_nurse_ward_policy: Authorisation_Policy

We then set the role of the policy to a staff nurse:

staff_nurse_ward_policy.role = staff_nurse

We can then assign the task and organisational domain to this policy:

staff_nurse_ward_policy.task = nurse_patient
staff_nurse_ward_policy.org_domain = ward

### 5.5.6 Verifying Policies

The final step is to verify policies through scenarios. In order to illustrate this we can define a scenario. We assume there is a nurse Judy_Smith, a staff nurse in ward_A. We therefore define these instances as follows:

Judy_Smith: Agent
ward_A: Org_Domain

The organisational domain instantiation relationship is defined as follows:

{ ward_A ↦ ward } ∈ insd

We then define an instantiated role to represent a staff nurse on ward_A:

staff_nurse_ward_A: Role
staff_nurse_ward_A.org_function = ward_nursing
staff_nurse_ward_A.org_domain = ward_A
staff_nurse_ward_A.authority = staff_nurse

We can now assign the role to Judy Smith:

{ Judy_Smith ↦ staff_nurse_ward_A } ∈ role_assignment

We define an instantiation of an asset nursing_record_1:

nursing_record_1: Asset
{ nursing_record_1 ↦ nursing_record } ∈ insa

The asset domain of the nursing_record_1 is assigned to the ward as follows:

{ nursing_record_1 ↦ ward_A } ∈ asset_domain

We then define the following instantiated task, that represents the action of reading the nursing record nursing_record_1:

read_nursing_record_1: Task
{ read_nursing_record_1 ↦ read_nursing_record } ∈ inst

This task relates to the asset nursing_record_1 in task_asset_dependency as follows:

{ nursing_record_1 ↦ nursing_record } ∈ task_asset_dependency

The scenario of Judy_Smith reading the nursing record nursing_record_1 can be defined as the following mapping between the user Judy_Smith and the corresponding instantiated task:

{ Judy_Smith ↦ read_nursing_record_1 } ∈ performs

If this is a valid performs definition, the following invariant that was introduced in section 5.4.7 must be maintained:

∀user: Agent; user_task: Task • user_task ∈ perfoms ( {user} ) ⇒
∃role: role_assignment ( {user} ) •
∃policy: Authorisation_Policy • policy.role ∈ inhr* ( insr ( {role} ) ) ∧
inst ( {user_task} ) ⊆ aggt* ( {policy.task} ) ∧
∀asset: task_asset_dependency ( {user_task} ) •
role.org_domain ∈ asset_domain ( {asset} )

In order to prove this we apply proof rules ( Woodcock & Davies, 1992). We need to check the invariant is satisfied for the agent Judy_Smith and the above defined user_task.

Applying the ∀ elimination rule twice, substituting Judy_Smith for user and read_nursing_record_1 for user_task, the expression user_task ∈ perfoms ( {user} ) is true because the following holds:

read_nursing_record_1 ∈ performs ( {Judy_Smith) } )

Applying the ⇒ elimination rule and the substitutions we applied above then the following expression must be true:

∃role: role_assignment ( {Judy_Smith} ) •
∃policy: Authorisation_Policy • policy.role ∈ inhr* ( insr ( {role} ) ) ∧
inst ( {read_nursing_record_1} ) ⊆ aggt* ( {policy.task} ) ∧
∀asset: task_asset_dependency ( {read_nursing_record_1} ) •
role.org_domain ∈ asset_domain ( {asset} )

After applying the ∃ elimination rule twice, substituting staff_nurse_ward_A for role, and staff_nurse_ward_policy for policy, we obtain the following expression that we need to prove is true:

staff_nurse_ward ∈ inhr* ( insr ( {staff_nurse_ward_A} ) ) ∧
inst ( {read_nursing_record_1 } ) ⊆ aggt* ( {nurse_patient} ) ∧
∀asset: task_asset_dependency ( {read_nursing_record_1} ) •
ward_A ∈ asset_domain ( {asset} )

The expression staff_nurse_ward ∈ inhr* ( insr ( {staff_nurse_ward_A} ) ) expands to staff_nurse_ward ∈ {staff_nurse_ward}, and therefore it is true. The expression

inst ( {read_nursing_record_1 } ) ⊆ aggt* ( {nurse_patient} )

expands to {read_nursing_record} ⊆ {nurse_patient, read_nursing_record, create_nursing_record}; this is true, so we can also eliminate it.

The expression task_asset_dependency ( {read_nursing_record_1} ) expands to {nursing_record_1}. After applying the ∀ elimination rule, and substituting nursing_record_1 for asset we obtain:

ward_A ∈ asset_domain ( {nursing_record_1} )

Finally by expanding the function mapping asset_domain ( {nursing_record_1} ) we get ward_A, so the expression is true. Thus the scenario is consistent with the policy definition.

## 5.6 Chapter Summary

In this chapter we have presented a framework for formally defining the organisational context. It comprises a meta-model and a set of heuristics. The meta-model is based on the principles by which organisations are differentiated on the basis of authority, functions and markets, enabling us to relate roles to the organisational context, and define access policies. A role is defined as a level of authority and organisational function within an organisational domain. We defined a set of heuristics for deriving definitions for the organisational context. We then explored how access policies can be defined to enforce the least minimum privileges principle. Finally we showed how we can verify that scenarios are consistent with policies.

# Chapter 6

# Extending the i* Framework and Formal Tropos

In the last chapter we presented a framework for developing access policies. What is unique about it compared to other approaches is that it allows us to define policies based on the characteristics by which an organisation divides work and assigns authority.

In this chapter we address the problem of extending current requirements modelling approaches and in particular how to integrate the conceptual framework that we presented in the previous chapter into an existing requirements modelling approach. There are two reasons for this: firstly, to be able to define access policies, independent of a specific system; and secondly to be able to define the constraints on functional requirements.

We demonstrate how formal Tropos and i* diagrams can be extended to incorporate the principles of the framework that we presented in the last chapter. In order to illustrate the extensions, we use the case study that we introduced in chapter 4.

## 6.1 Extensions to Tropos

In this section we describe the extensions to Tropos. We have already presented the definition of a role, which we associated with the organisational contextual

elements of authority, organisational function, and organisational domain. We now demonstrate how formal Tropos can be extended to include the meta-concepts of the framework, how domain modelling can be carried out, and how instantiation can be achieved in order to verify policies.

### 6.1.1 Representing Strategic Rational Diagrams in Formal Tropos

Fuxman *et al.* (2001) do not explain how a Strategic Rationale (SR) model in i* can be represented in formal Tropos. As we described in chapter 4, an SR model is essential to the definition of a policy in that the actor boundary is used to define an access policy to the tasks and resources within the boundary, and hence we extend formal Tropos accordingly.

Our representation of an SR diagram is illustrated below, using an indentation to represent the means-end to a goal, and the resource dependency relationship to the actor. We have added a type attribute to actor to enable us to differentiate between agents, positions, and roles.

**Actor** Programmer
  **Type** Position
  **Goal** Develop Software
    **Mode** Achieve
    **Task** Write Code
      **Resource** Program Module

We have also added inheritance, aggregation, and instantiation between domain elements using the keywords **IsA**, **Part**, and **INS** respectively, as used in i*.

### 6.1.2 Linking Actor Definitions to the Organisational Context

Next we consider the modelling of roles and associated organisational characteristics. Referring back to the example of Liu *et al.* in chapter 2, family doctor was defined as an agent and Dr. Anthony as an instantiation of that agent. We follow

the convention of i* in that an actor of type Agent maps onto an agent in our framework, since this represents a physical agent. For representing abstract and instantiated roles we have decided to use actor of type role in the i* framework.

The organisational context definitions, authority, organisational function, and organisational domain can be defined as classes. In the last chapter we defined a role as being associated with these three organisational contextual characteristics. Although the i* framework differentiates actor definitions further into agents, roles and positions, formal Tropos only includes actor definitions. We have added a type definition to the actor definition to differentiate between roles, positions, and agents.

The following are definitions of the software project organisation we introduced in chapter 4, to illustrate how to link actor definitions to the organisational context. First of all we define two levels of authority, Project Manager and Engineer. A Project Manager is senior to an Engineer.

**Authority** Project Manager

**Authority** Engineer
  **Senior** Project Manager

We then define some organisational functions. In the organisational functions below, IT Testing is a specialisation of IT Development.

**Organisational Function** IT Development

**Organisational Function** IT Testing
  **IsA** IT Development

There are two organisational domains of Project and Sub-Project.

**Organisational Domain** Project

**Organisational Domain** Sub-Project
  **Part** Project

With these organisational contextual definitions, we can now define the following actors:

**Actor** IT Project Manager
**Type** Role
**Authority** Project Manager
**Organisational Function** IT Development
**Organisational Domain** Project

**Actor** Test Engineer
**Type** Role
**Authority** Engineer
**Organisational Function** IT Test
**Organisational Domain** Project

The properties we described in the last chapter of inheritance and aggregation can modelled through a **Part** and **IsA** characteristics respectively, these correspond to the inheritance and aggregation mappings, which we introduced in the previous chapter, as follows:

The relation inhf representing the inheritance between organisational functions is mapped on to the **IsA** attribute of an organisational function definition, where the organisational function that has the attribute, is inheriting from the organisational function that is the attribute.

The aggregation between domains aggd is represented by the **Part** attribute of organisational domain, where the organisational domain that has the attribute, is a sub-domain of the organisational domain that is defined as the attribute. So for example the organisational domain Sub-Project is a sub-domain of Project, which is related in the form of a **Part** attribute for Sub-Project domain. The **IsA** characteristic can be used to define the inheritance between the organisational functions of IT Development and IT Testing.

In the last chapter we described an invariant which relates the inheritance between roles and those between organisational functions and organisational domains. This invariant also holds here, whereby, for an actor, whether a position or role, which has an organisational function, which is inherited from an organisational function of

another actor with the same organisational domain means an inheritance exists between those two actors.

### 6.1.3 Tasks and Resource Definitions

The task and asset definitions in the framework meta-model presented in chapter 5 correspond to task and resource definitions in Tropos. The task aggregation aggt is mapped into the i* framework as a task decomposition link, which is not currently defined in formal Tropos. The link between tasks and assets in the relation, task_asset_dependency, also corresponds to a decomposition link between tasks and resources in the i* framework. This is simply represented through defining the resources associated with a task underneath a task definition but indented. Task decomposition is represented in an identical way.

The following task Test Software can be divided into three sub-tasks of Prepare Test Plan, Read Test Plan, and Update Test Result, and these subtasks in themselves depend on the resources Test Plan and Test Result:

**Task** Test Software
  **Task** Prepare Test Plan, Read Test Plan, Update Test Result

**Task** Prepare Test Plan
  **Resource** Test Plan

**Task** Read Test Plan
  **Resource** Test Plan

**Task** Update Test Result
  **Resource** Test Result

The following task Approve Software Release depends on the resources Test Result and Release Note:

**Task** Approve Software Release
  **Resource** Test Result
  **Resource** Release Note

### 6.1.4 Defining Access Policies

Access policies as we defined in the last section are effectively mapped to actors and their boundaries. In the i* framework these are represented by the tasks and associated resources within the actor boundary.

Effectively an access policy is defined through the actor boundary. So for example we may want to restrict the task Approve Software Release to the IT Project Manager. The above policy definition is simply represented as an actor boundary.

We can therefore define the following actor:

**Actor** IT Project Manager
**Type** Role
**Authority** Project Manager
**Organisational Function** IT Development
**Organisational Domain** Project
**Task** Approve Software Release

### 6.1.5 Defining Scenarios

An agent in the framework meta-model presented in chapter 5 corresponds to an instantiated agent in Tropos, and the role corresponds to a Tropos role definition. An instantiated role in extended Tropos is associated with an instantiated organisational domain. Continuing our example of a software project organisation, and focusing on the policy to approve a software release, we can define an organisational domain as follows:

**Organisational Domain** Library Administration System Project **INS** Project

Since this policy involves access to the assets, Release Note and Test Result, through the task Approve Software Release, we also need an instantiation for the corresponding task and resources:

**Resource** Release Note Version 1 **INS** Release Note

**Resource** Test Result Version 1 **INS** Test Result

**Task** Approve Software Release Version 1 **INS** Approve Software Release
**Resource** Release Note Version 1
**Resource** Test Result Version 1

Having defined these instantiations we can then define the instantiation of the role:

**Actor** IT Project Manager Library Admin. Project **INS** IT Project Manager
**Type** Role
**Organisational Domain** Library Administration System Project

The definition of an individual executing a task is as follows, assigning an agent to an instantiated role through the relation **OCCUPIES**, and relating it to a task:

**Actor** John Smith **OCCUPIES** IT Project Manager Library Admin. Project
**Type** Agent
**Task** Approve Software Release Version 1

This instantiation represents a combination of the agent role assignment (role_assignment), and agent task mapping (performs) in the framework presented in chapter 5.

## 6.2 Representing the Organisational Context in i*

In this section, we propose how the formal Tropos extensions we introduced in the previous section could be represented in i* framework diagrams.

As explained earlier in this chapter, there are two types of i* framework diagrams: Strategic Dependency (SD) diagrams that show the dependency between actors, and Strategic Rationale (SR) diagrams that focus on a single actor and his goals.

An example of an SD diagram is shown in figure 6.1. In order to represent the organisational context on SD diagrams we have introduced a new symbol to represent organisational domains, a dashed circle with a label containing the name.

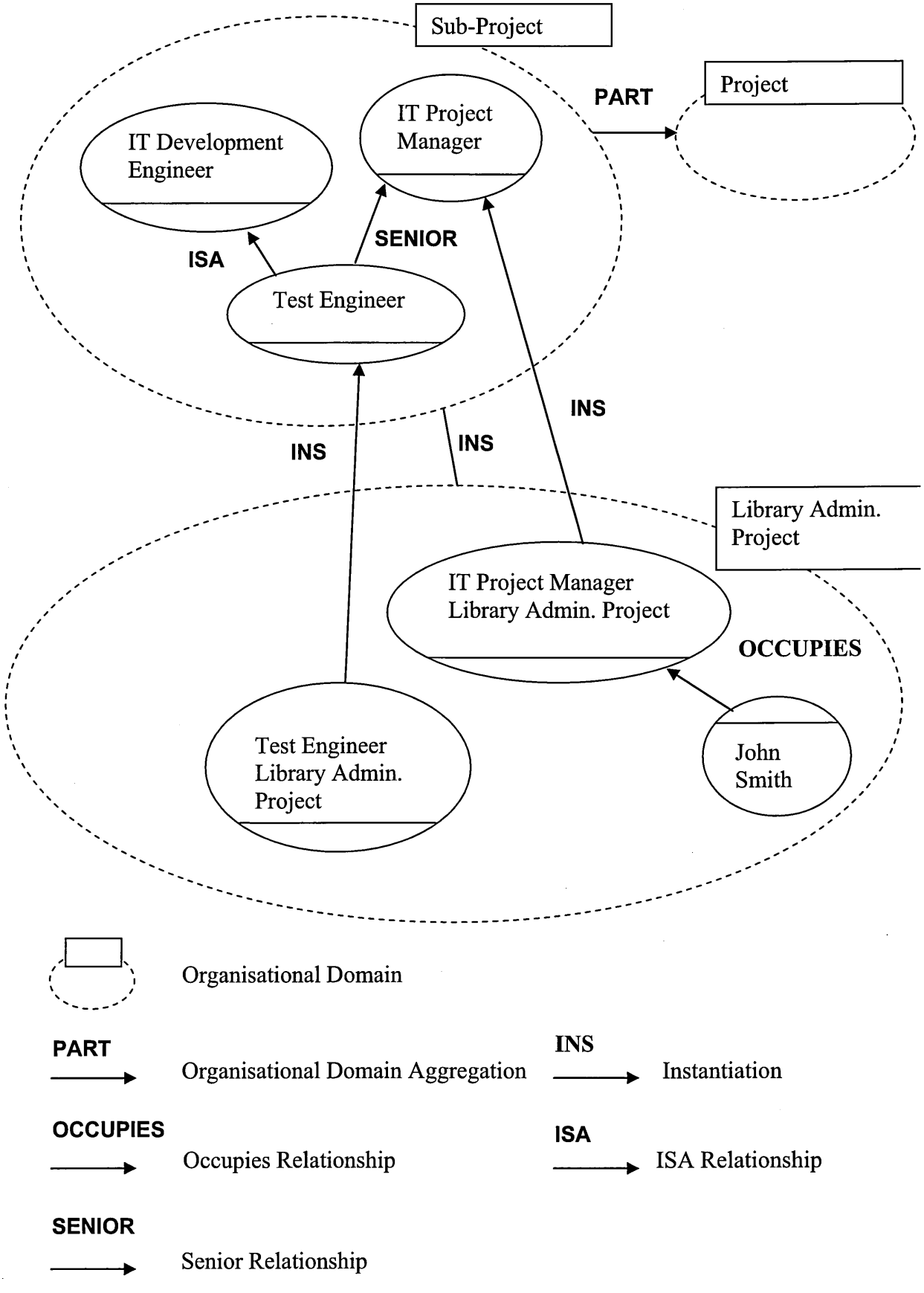



**Figure 6.1 Extended i* Strategic Dependency Diagram**

The seniority relation is simply represented by an arrow with the keyword **SENIOR** between actors. This shows that the actor in this relationship, to which the arrow is pointing, has a level of authority more senior to the other actor for the same function and within the same organisational domain. The other relations and symbols already exist in the i* framework and have been reused.

We have only represented those aspects that are relevant to relating actors to the organisational context. Other aspects of an SD diagram would be modelled as before. The organisational function and level of authority could be represented on the SR diagram as labels.

Diagrams of the organisational structure could be represented independently of actors, to show how organisational domains, organisational functions, and levels of authority relate to one another. We do not explore this however in this thesis.

## 6.3 Mapping Formal Tropos Policies to Framework Definitions in Z

The extensions to formal Tropos that we have introduced in this chapter are derived from the framework meta-model that we introduced in chapter 5. In order to apply the approach to analysis that we presented in the previous chapter, we need to be able to translate the formal Tropos model into the equivalent constructs in Z. Table 6.1 summarises translation rules to be able to do this.

| **Formal Tropos** | **Framework Definitions in Z** |
|---|---|
| **Meta-Concept Translation Rules** | |
| **Task** T | T: Task |
| **Resource** R | R: Asset |
| **Authority** A | A: Authority |

| | |
|---|---|
| **Organisational Domain** OD | OD: Org_Domain |
| **Organisational Function** OF | OF: Org_Function |
| **Actor** AC<br>**Type** Role<br>**Authority** AU<br>**Organisational Function** OF<br>**Organisational Domain** OD | AC: Role<br>AC.authority = AU<br>AC.org_function = OF<br>AC.org_domain = OD |
| **Authority, Inheritance and Aggregation Hierarchies Translation Rules** | |
| **Organisational Function** OF1<br>**IsA** OF2 | OF1: Org_Function<br>{ OF1 ↦ OF2 } ∈ inhf |
| **Organisational Domain** OD1<br>**Part** OD2 | OD1: Org_Domain<br>{OD1 ↦ OD2} ∈ aggd |
| **Task** T1<br>**Task** T2<br>**Task** T3<br>:<br>**Task** TN | T1: Task<br>{ T1 ↦ T2, T1 ↦ T3, ...T1 ↦ TN } ∈ aggt |
| **Task** T<br>**Resource** R1<br>**Resource** R2<br>:<br>**Resource** RN | T: Task<br>{ T ↦ R1, T ↦ R2, ... T ↦ RN } ∈<br>task_asset_dependency |
| **Actor** AC1 **ISA** AC2 | AC1: Role<br>{ AC1 ↦ AC2 } ∈ inhr |
| **Policy Definition Translation Rule** | |
| **Actor** A<br>**Type** Role<br>**Task** T | ATP: Authorisation_Policy<br>ATP.role = A<br>ATP.task = T |
| **Instantiation Translation Rules** | |
| **Resource** A1 **INS** A2<br>**Organisation Domain** OD1 | A1: Asset<br>{ A1 ↦ A2 } ∈ insa<br>{ A1 ↦ OD1 } ∈ asset_domain |
| **Task** T1 **INS** T2<br>**Resource** R1<br>**Resource** R2<br>:<br>**Resource** RN | T1: Task<br>{ T1 ↦ T2 } ∈ inst<br>{ T1 ↦ R1, T1 ↦ R2,... T1 ↦ RN } ∈<br>task_asset_dependency |

| | |
|---|---|
| **Organisational Domain** OD1 **INS** OD2 | OD1: Org_Domain<br>{ OD1 ↦ OD2 } ∈ insd |
| **Actor** ACOD1 **INS** AC<br>**Type** Role<br>**Organisation Domain** OD1 | ACOD1: Role<br>{ ACOD1 ↦ AC } ∈ insr<br>ACOD1.org_domain = OD1 |
| **Actor** AG **OCCUPIES** ACOD1<br>**Type** Agent<br>**Task** T | AG: Agent<br>{ AG ↦ ACOD1 } ∈ role_assignment<br>{ AG ↦ T } ∈ performs |

**Table 6.1 Mapping Formal Tropos to Framework Definitions in Z**

We can illustrate the mapping rules with some examples. The definition for the level of authority of Project Manager maps using the corresponding meta-concept translation mapping rule as follows:

**Authority** Project Manager

Project Manager: Authority

The definition for the level of authority Engineer, which has a seniority relationship with Project Manager, is mapped using the authority hierarchy mapping rule. This rule is an extension of the meta-concept translation rule used above. The mapping is as follows:

**Authority** Engineer
  **Senior** Project Manager

Engineer: Authority
{ Engineer ↦ Project Manager } ∈ senior

The actor definition IT Project Manager is mapped as follows:

**Actor** IT Project Manager
  **Type** Role
  **Authority** Project Manager
  **Organisational Function** IT Development
  **Organisational Domain** Project

IT Project Manager: Role
IT Project Manager.authority = Project Manager
IT Project Manager.org_function = IT Development
IT Project Manager.org_domain = Project

The definition for the agent John Smith maps as follows:

**Actor** John Smith **OCCUPIES** IT Project Manager Library Admin. Project
**Type** Agent
**Task** Approve Software Release Version 1

John Smith: Agent
{ John Smith ↦ IT Project Manager Library Admin. Project } ∈ role_assignment
{ John Smith ↦ Approve Software Release Version 1 } ∈ performs

The prerequisite for this rule is that the definitions for the role IT Project Manager Library Admin. Project, the instantatiated task Approve Software Release Version 1, and associated resources, Release Note Version 1 and Test Results Version 1, have been defined.

## 6.4 Chapter Summary

In this chapter we have extended the requirements modelling language of formal Tropos, and demonstrated how these extensions could be represented on i* SD diagrams. We selected the i* framework for extension because of its focus on the social context and how actors relate to one another, which is the basis for the organisational context.

We demonstrated how the framework we presented in chapter 5 can be used as a basis for extending the i* framework. It addresses the problems that we identified in chapter 3 with regard to the actor definitions. In chapter 3 we highlighted the need to have precise actor definitions as a prerequisite for defining policies, and in this chapter we demonstrated how actor definitions in formal Tropos, an extension of i*, can be extended to link them to the organisational context.

We also showed how formal Tropos definitions can be mapped to the Z model presented in chapter 5, and hence be used as a basis for formal reasoning.

# Chapter 7

# Automated Analysis using Alloy

In chapter 5 we presented a framework for defining policies and scenarios in Z, and reasoning about them in order to verify that the scenarios are consistent with the policies defined.

In this chapter we propose an automated approach using the modelling and analysis tool, Alloy. We first justify the use of Alloy for this purpose, and introduce the Alloy modelling language. We then illustrate how we can translate the framework meta-model introduced in chapter 5 into the Alloy modelling language. Finally, having translated the meta-model and access policy constructs from our meta-model into Alloy, we demonstrate how the verification of policies can be carried out.

## 7.1 Verification Alternatives

Two alternatives for automating the analysis were considered.

One alternative would be to use a theorem prover. There are tools which would enable us to perform proofs in the Z language, examples of which include Z-Eves or CadiZ. However, theorem provers are difficult to use as expert knowledge in logic and set theory is required to be able to define a proof strategy.

A more promising alternative to formal proofs is the use of lightweight formal checking tools, so called because they check formal models without proving theorems, and hence requiring less expertise in logic and set theoretics than is

necessary for theorem provers. They can be used for validating the model as they are supported by tools. There are essentially those tools that enable a specification to be animated, such as IFAD (Fitzgerald & Larsen, 1998), and those that perform exhaustive checks to determine whether assertions are adhered to, such as Alloy (Jackson, 2004) or NuSMV (Cimatti *et al.*, 2002). The checker tools are much more rigorous than animation tools, as through an animation only limited scenarios can be tested, whereas in the case of an exhaustive checker many more scenarios will be tested. The NuSMV checking tool has been applied in the case of formal Tropos (Fuxman *et al.*, 2001). An advantage that Alloy has over NuSMV for this research project, is that it is based on the Z language, employing the same logical and set theoretic notions as Z, and hence is much easier to translate to than an NuSMV model would be. An advantage that NuSMV has, is that it can evaluate some temporal constraints, though not all. For the purposes of using NuSMV for evaluating formal Tropos, Fuxman *et al.* (2001) extended the tool to handle additional forms of temporal constraints, and also to be able to generate instances automatically. Since the security constraints that we are exploring are always to be maintained, the temporal constraints are not interesting and were therefore not a decisive factor in selecting a tool for validation. Due to the similarities of Z to Alloy and the ease of translation, Alloy was chosen. We used Alloy version 3.0 (Jackson, 2004).

## 7.2 Introduction to Alloy

The Alloy language is supported by an Alloy analyser. As we mentioned above the Alloy language can be viewed as a subset of Z, Alloy is a declarative language which enables the structural properties and functions of a system to be modelled. Assertions can be defined representing properties of the system that must be adhered

to. The tool enables one to execute functions and so animate a model and also check assertions hold by searching for counter examples.

### 7.2.1 Types and Relations

Fundamentally Alloy models are constructed from relations and atoms. Atoms are entities, and relations are mappings between different types of atoms. In Alloy all expressions pertain to relations and sets do not exist; they are in effect represented by unary relations.

Basic types in Z can be translated into signatures in Alloy using the keyword sig. A signature in Alloy represents a type of atom. So for example sig Org_Domain {} defines the basic type of an organisational domain. An instantiation of this type would result in an atom. The instantiation of atoms is carried out in Alloy when a model assertion or function is executed.

Relations between atoms can be defined within the signatures; for example if we wish to define a type asset, which is related to organisational domain, this can be defined as a field within the signature of type asset as follows:

```
sig Asset {
asset_domain: Org_Domain }
```

This can be used as a correspondence to the following definition in Z.

asset_domain: Asset → Org_Domain

A field can also be defined as a relation. We could therefore define the above asset domain relation as follows:

```
sig Asset_Domain {
asset_domain: Asset->Org_Domain }
```

However defining it in this way would result in additional definitions of type Asset_Domain, which are unnecessary.

Alloy enables us to define sub-signatures, which effectively represent subsets of a specific signature, this is achieved using an extends declaration as follows:

```
medical_record extends Asset {}
```

This is useful for defining derived types such as in this case where a medical record is a type of asset. Sub-signatures automatically inherit the properties of the signature that they extend. A signature can be defined as one, which means that the signature contains only a single atom. This is useful for modelling instantiations of signature types. For example we can define some specific agents as follows:

```
one sig John, James, Fred, Jonathan extends Agent {}
```

In Z this could be equivalently defined as follows:

```
John, James, Fred, Jonathan: Agent
```

There are three special set operators iden, univ and none, which represent the identity relation, which includes relations of each element to itself, the universal relation, which includes all elements, and the empty relation, which contains no element.

### 7.2.2 Operators and Quantifiers

Types as described in the last section represent sets of atoms; the following set operators are available:

+ union

& intersection

– difference

For comparison there are the operators:

= equivalence

in membership

So for example if nurses and doctors are sets of agents and hospital_staff is a union of nurses and doctors then the following apply:

hospital_staff = nurses + doctors

nurses = hospital_staff - doctors

In addition there are logical operators:

| | |
|---|---|
| ! | negation operator, whereby !A is not A. |
| && | conjunction operator, whereby A && B is A and B. |
| \|\| | disjunction operator , whereby A \|\| B is A or B. |
| => | the implies operator, whereby A => B means if A is true then B is also true. |
| <=> | the bi-implies operator, whereby A <=> B means A implies B and vice-versa. |

The quantifiers are:

| | |
|---|---|
| all A: B \| C | is the universal quantifier, for all A in B, C is true. |
| some A: B \| C | is the existential quantifier, for some A in B, C is true. |
| sole A: B \| C | represents that no more than one A in B exists for which C is true. |
| no A: B \| C | represents that no A in B exist for which C is true. |
| one A: B \| C | represents that exactly one A exists in B for which C is true. |

Relational operators are:

| | |
|---|---|
| . | is a join between two relations. For the join p.q, it is the relation arising by taking every combination of each tuple in p and q, and including their join. If p is a set and q is a binary relation, then this produces the relational image of p under q. |
| ~ | is the transpose operator, which reverses all the tuples in the relation. |
| ^ | is the transitive closure operator. If a signature A { f: set A } contains a relation to elements of the same type then a transitive closure of an element x of type A contains all x.f + x.f.f + x.f.f.f... |

| | |
|---|---|
| * | is the reflexive closure operator, which contains the transitive closure plus the relationship of an element to itself. If a signature A { f: set A } contains a relation to elements of the same type then a reflexive closure of an element x of type A contains all x + x.f + x.f.f + x.f.f.f... |
| -> | is the product of two relations. The product p->q is the relation given by taking every combination of each tuple in p and q, and concatenating them. |

### 7.2.3 Invariant and Function Definitions

Invariants in the model can be defined as facts. For example the following definition of a role has a relation insr, which relates an instantiated role to an abstract role.

```
sig Role {
insr: set Role }

fact {
all role1, role2: Role | role1.insr = role2 => role2.insr = none }
```

This fact restricts the definitions of roles such that a role can not be instantiated from a role that is itself an instantiated role.

Predicates can be defined which describe how state changes can be enacted. Predicate definitions include parameters, and describe how the state of these parameters are changed. For example the following predicate describes how an organisational domain can be added to the aggregation of another organisational domain.

```
pred add_agg_domain (od1, od1', od2: Org_Domain) {
od1'.aggd = od1.aggd  + od2 }
```

### 7.2.4 Recursive Relations

As we mentioned above, a relation in Alloy can be defined as a field of a signature. If it is of the same type then we can define a recursive relation, which is useful for representing hierarchies. Thus the following is how we define an organisational domain and the aggregation hierarchy as a relation:

```
sig Org_Domain {
aggd: set Org_Domain }
```

This is equivalent to the following definitions in Z:

```
[Org_Domain]
aggd: Org_Domain ⇸ Org_Domain
```

In defining invariants or assertions we sometimes need to define the transitive or transitive-reflexive closure of a relation. An example of this is the following constraint that we defined for an organisational domain, that an organisational domain can not be an aggregation of itself:

$\forall od: Org\_Domain \bullet od \notin aggd^{+} ( \{od\} )$

This constraint can be defined in Alloy as follows:

```
fact { all od: Org_Domain | od !in od.^aggd }
```

The expression $aggd^{+} ( \{od\} )$ is translated into Alloy as od.^aggd. The join operator acts as a relational image of od under the transitive closure of the relation aggd, restricting the domain to od.

Although aggd is a field of Org_Domain, it can be referenced without being on the left hand side of a join operator; the expression od.aggd gives us a set of type Org_Domain, but aggd is the binary relation of type Org_Domain ->Org_Domain. For example the above constraint could be defined as follows:

```
fact { all od: Org_Domain | od->od !in ^aggd }
```

### 7.2.5 Modules

Models can be divided into modules. Models or parts of models can therefore be reused by defining them within a module that can then be included by other models. There are two keywords *open* and *use* to include modules, the only difference being that when *use* is used to include modules definitions, then they have to be qualified,

whereas with *open* they do not. Thus in the following example the signature Org_Domain we defined above could be included in the module Organisation:

```
Module Organisation
sig Org_Domain {
.. }
```

This can then be included in a module that defines the predicate add_agg_domain as follows:

```
Module Organisation_Functions
open Organisation
pred add_agg_domain (od1, od1', od2: Org_Domain) {
od1'.aggd = od1.aggd + od2 }
```

## 7.3 Modelling Policies in Alloy

We now demonstrate how the policy framework meta-model presented in chapter 5 can be represented in the Alloy language, and how analysis can be performed.

### 7.3.1 Modelling the Framework Meta-Concepts and Relations

As we explained in chapter 5, the policy framework consists of a meta-model with meta-concepts and their relations. It is these meta-concepts and relations that form the basis of the extensions to formal Tropos. We can now demonstrate how these meta-concepts and relations translate from our Z definitions into Alloy. A diagram of the meta-model in Alloy is shown in figure 7.1.

Meta-concepts can be represented as signatures in Alloy so for example, the meta concept [Org_Domain] translates simply into the following signature:

```
sig Org_Domain {}
```

A similar translation applies to the other meta concepts of [Asset], [Task], [Authority], and [Org_Function].

Inheritance and aggregation hierarchies in our Z model that are modelled as relations, are also represented as relations in Alloy; but as described above, relations

must be defined within signatures; thus for example, in order to represent the relation aggd: Org_domain ⇸ Org_domain the Org_Domain signature is extended as follows:

```
sig Org_Domain {
aggd: set Org_Domain }
```

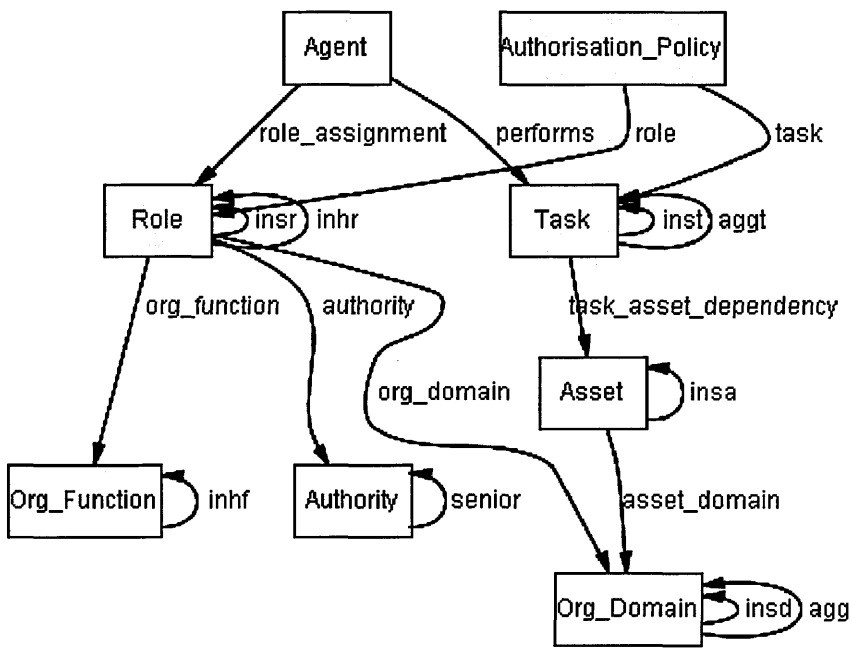



**Figure 7.1 Framework Meta-Model in Alloy**

Roles and policies are composite meta-concepts, and include other meta-concepts as members. So, for example the role is defined as follows:

```
sig Role {
authority: Authority,
org_function: Org_Function,
org_domain: Org_Domain,
inhr: set Role }
```

As we see here there are three members of type Authority, Org_Function and Org_Domain, and in addition there is the role inheritance hierarchy, defined using the relation inhr.

### 7.3.2 Policy Domain Definitions

In order to define the model for an application domain, specific definitions need to be made. These are achieved by creating unique atoms as subsets of the meta-concepts. So for example if we wish to define ward as an organisational domain, with hospital as an aggregation, we would define it as follows:

```
one sig ward extends Org_Domain { }
fact { ward.aggd = hospital }
```

All other domain specific definitions are made using this form of construct. If a relation such as aggd does not relate to a set, then the range is defined as none, as in the following example:

```
one sig hospital extends Org_Domain { }
fact { hospital.aggd = none }
```

This is necessary otherwise the Alloy tool may set the range to an arbitrary value.

### 7.3.3 Policy Framework Domain Instantiations

In order to instantiate domain concepts we need use the instantiation relations that we translate from our framework meta-model in Z. So for example the Org_Domain signature includes an instantiation relation for this purpose as follows:

```
sig Org_Domain {
insd: set Org_Domain }
```

This represents the relation:

insd: Org_domain ⇸ Org_domain.

This form of instantiation also follows for tasks, assets and roles. Agents are themselves instantiations but are assigned to instantiated roles. The following shows an example of an agent definition that has been assigned the instantiated role of a General Practitioner.

```
one sig Dr_Smith extends Agent {}
fact { Dr_Smith.role_assignment = General_Practitioner_Dr_Jones_Practice }
```

Thus if we define an instantiation of a General Practice, then as above with the policy domain definitions we create unique atoms as subsets of the meta-concepts, but relate them to the domain concepts, through the instantiation relations, defined as a fact. So if Dr Jones Practice is an instantiation of General Practice, this is defined as follows:

```
one sig Dr_Jones_Practice extends Org_Domain {}
fact { Dr_Jones_Practice.insd = General_Practice }
```

As with policy domain definitions, for empty relations we set the range to none.

### 7.3.4 Policy Verification

In order to verify policies in our framework as we explained in chapter 5, we use an instantiated task that represents the carrying out of a task on specific instances of assets, related through the task asset dependency. In Alloy we represent this via a performs relation, which defines an agent performing a task, and is defined as a relation on an agent through the agent signature as follows:

```
sig Agent {
.
.
performs: set Task }
```

The policy framework invariant that we described in chapter 5 that needs to be satisfied in order for instantiations of performs relations to satisfy policies, is as follows:

```
fact {
all user: Agent, task: Task | user_task in user.performs =>
some user_role: user.role_assignment | some policy: Authorisation_Policy |
policy.role in user_role.insr.*inhr
&& task.inst in policy.task.*aggt
&& all asset: user_task.task_asset_dependency |
user_role.org_domain in asset.asset_domain }
```

As described above consistency checks can be made through the use of assertions. In order to check a particular scenario an assertion can be defined. The tool will then search for a solution that breaks the assertion; if no solution can be found, then with a high degree of certainty we can assume that the assertion is correct.

### 7.3.5 Model Consistency Checks

In chapter 5 we described a number invariants that ensure the model is consistent with principles of the framework. These include invariants such as ensuring that roles do not inherit themselves or that instantiated elements are not used in policy definitions. There are two alternatives as to how this can be checked in Alloy: the first is to define them as facts; the second is to define them as assertions. The first alternative is suitable when Alloy generates all the instantiations, as it will ensure that the invariants are enforced. Our method of instantiation, however, is to generate them explicitly to create specific scenarios. If a mistake is made in any of the domain and scenario definitions, then running a check in a policy verification may give no solution, because the invariants have not been satisfied, and not because the policy has been correctly verified. Checking assertions that the invariants have been satisfied will give us much greater confidence that the definitions have been made as intended. For example to ensure that roles do not inherit themselves we can run the following assertion:

```
assert check_role_inheritance {
all r: Role | r !in r.^inhr }
```

In chapter 5 we introduced invariants that need to be maintained if policy and scenario definitions are consistent. These are summarised in table 7.1.

| Alloy Invariant | Description |
|---|---|
| all r: Role \| r !in r.^inhr | Defines a role can not inherit itself. |
| all of: Org_Function \| of !in of.^inhf | Defines an organisational function can not inherit itself. |
| all od: Org_Domain \| od !in od.^aggd | Defines an organisational domain can not be an aggregation of itself. |
| all role1, role2: Role \| role2 in role1.^inhr => role2.org_function in role1.org_function.^inhf && role1.org_domain = role2.org_domain && role1.authority = role2.authority | Defines that a role, role2, that is inherited by role1 must have an organisational function that is inherited by the organisational function assigned to role1 and must have an identical level of authority and organisational domain. |
| no role: Role \| role.insr = none && role.org_domain.insd != none \|\| role.insr != none && role.org_domain.insd = none | This defines that an abstract role (i.e. non-instantiated) is not associated with an instantiated organisational domain, and that an instantiated role is not associated with an abstract domain. |
| all au: Authority \| au !in au.^senior | Defines that a level of authority can not be senior to itself. |
| all t: Task \| t !in t.^aggt | Defines that a task can not be an aggregation of itself. |
| all a: Agent \| all task: agent.performs \| task.inst != none | Defines that all tasks performed by an agent are instantiated. |
| all a: Agent \| all task: agent.performs \| all ins_asset: task.task_asset_dependency \| some asset: task.inst.task_asset_dependency \| asset in ins.asset.insa | Defines that all assets in the task asset dependency of a task performed by an agent are instantiated from assets in the task asset dependency of the corresponding task from which the task was instantiated. |
| all role1, role2: Role \| role1.insr = role2 => role2.insr = none | Defines that a role can not be instantiated from a role that itself is an instantiation. |

| | |
|---|---|
| all od1, od2: Org_Domain \| od1.insd = od2 => od2.insd = none | Defines that an organisational domain can not be instantiated from an organisational domain that itself is an instantiation. |
| all od1, od2: Org_Domain \| od1.aggd = od2 => od1.insd = none && od2.insd = none \|\| od1.insd != none && od2.insd != none | Defines that both organisational domains in an aggregation relation should be both either instantiated or non-instantiated. |
| all a1, a2: Asset \| a1.insa = a2 => a2.insa = none | Defines that an asset can not be instantiated from an asset that itself is an instantiation. |
| no policy: Authorisation_Policy \| policy.role.insr != none | Defines there is no policy associated with an instantiated role. |

**Table 7.1 Framework Invariants in Alloy**

### 7.3.6 Mapping from Formal Tropos into Framework Definitions in Alloy

In the previous chapter we presented a mapping from formal Tropos into the framework definitions in Z. In order to translate policies defined using formal Tropos into Alloy to carry out an analysis, we need to adapt these mapping rules. These are summarised in the table 7.2.

| **Formal Tropos** | **Meta-Model Definitions in Alloy** |
|---|---|
| **Meta-Concept Translation Rules** | |
| **Task** T | one sig T extends Task {} |
| **Resource** R | one sig R extends Asset {} |
| **Authority** A | one sig A extends Authority {} |
| **Organisational Domain** O | one sig O extends Org_Domain {} |
| **Organisational Function** OF | one sig OF extends Org_Function {} |

| | |
|---|---|
| **Actor** AC<br>**Type** Role<br>**Authority** AU<br>**Organisational Function** OF<br>**Organisational Domain** OD | one sig AC extends Role {}<br>fact { AC.authority = AU }<br>fact { AC.org_function = OF }<br>fact { AC.org_domain = OD } |
| **Authority, Inheritance and Aggregation Hierarchies Translation Rules** | |
| **Authority** AU1<br>**Senior** AU2 | one sig AU1 extends Authority {}<br>fact { AU1.senior = AU2 } |
| **Organisational Function** OF1<br>**IsA** OF2 | one sig OF1 extends Org_Function {}<br>fact { OF1.inhr = OF2 } |
| **Organisational Domain** OD1<br>**Part** OD2 | one sig OD1 extends Org_Domain {}<br>fact { OD1.aggd = OD2 } |
| **Task** T1<br>**Task** T2<br>:<br>**Task** TN | one sig T1 extends Task {}<br>fact { T1.aggt = T2 ...+TN } |
| **Task** T<br>**Resource** R1<br>:<br>**Resource** RN | one sig T extends Task {}<br>fact { T.task_asset_dependency = R1...+ RN } |
| **Actor** AC1 **ISA** AC2 | one sig AC1 extends Role {}<br>fact { AC1.inhr = AC2 } |
| **Policy Definition Translation Rule** | |
| **Actor** A<br>**Type** Role<br>**Task** T | one sig AP extends Authorisation_Policy {}<br>fact { AP.role = A }<br>fact { AP.task = T } |
| **Instantiation Translation Rules** | |
| **Resource** R1 **INS** R2<br>**Organisation Domain** OD1 | one sig R1 extends Asset {}<br>fact { R1.insa = R2 }<br>fact { R1.asset_domain = OD1 } |
| **Task** T1 **INS** T2<br>**Resource** R1 | one sig T1 extends Task {}<br>fact { T1.inst = T2 }<br>fact { T1.asset_dependency = R1 } |
| **Organisational Domain** OD1 **INS** OD2 | one sig OD1 extends Org_Domain {}<br>fact { OD1.insd = OD2 } |

| | |
|---|---|
| **Actor** ACOD1 **INS** AC<br>**Organisation Domain** OD1 | one sig ACOD1 extends Role {}<br>fact { ACOD1.insd = AC }<br>fact { ACOD1.org_domain = OD1 } |
| **Actor** AG **OCCUPIES** ACOD1<br>**Task** T1 | one sig AG extends Agent {}<br>fact { AG.role_assignment = ACOD1}<br>fact { AG.performs = T1 } |

**Table 7.2 Mapping Framework Z Definitions to Alloy**

### 7.3.7 Structuring Modules

As we described above, Alloy enables the model to be divided up; this helps in scaling the model for large applications as the model can be broken down into manageable chunks. Figure 7.2 shows how the framework definitions represented in an Alloy model can be divided into modules.

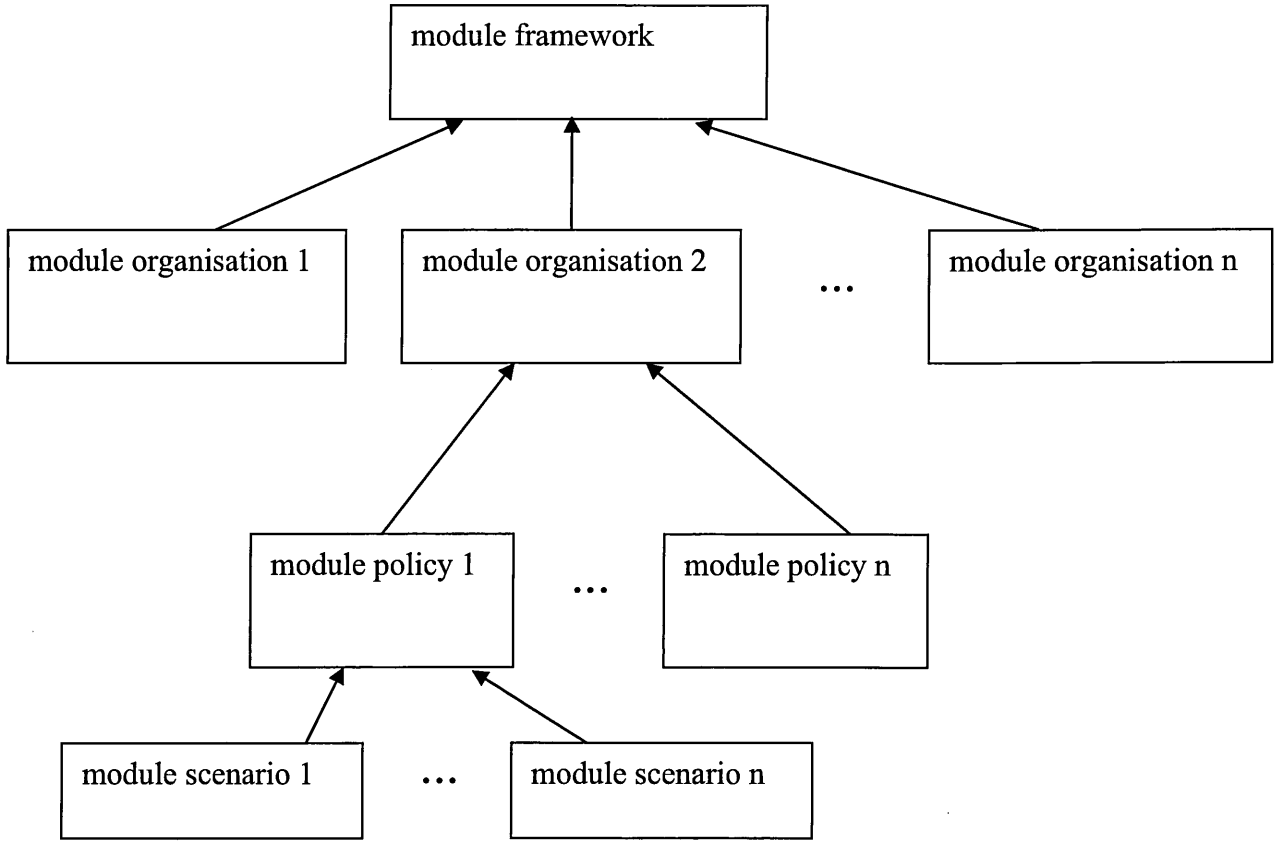



**Figure 7.2 Module Structure of the Framework in Alloy**

At the highest level is the module framework that includes all the meta-definitions. An organisation can be divided up, so it is possible to split the organisational contextual and role definitions into separate modules for parts of the organisation. So, for example, the organisational contextual and role definitions for the IT development department of a bank could be separately defined from those in the bank branches. These definitions are then included in modules organisation 1 to n for n organisational units. Then for each of these organisational units, each policy can be defined in its own policy module. Finally for each of these policies, several scenarios can be defined, each in its own module.

## 7.4 Alloy Evaluation

The key advantage of Alloy is the automated checking that the tool performs. However it does not do this by proving the assertion as we demonstrated using Z in chapter 5, but through a search for counter examples. If a counter example is not found, this does not necessarily mean that the model is consistent or correct, it can also produce this result if the model is inconsistent. It is a problem that the tool does not display the reasoning. However by negating assertions it is possible to produce counter examples, as we demonstrated. Examining counter examples, which the tool displays, gives us greater confidence that a model we create is correct. Executing assertions to check the invariants that we presented in table 7.1, are also very useful in identifying inconsistencies.

## 7.5 Chapter Summary

In this chapter we have demonstrated how automated analysis can be carried out using the tool Alloy. We began by outlining the reasons for using Alloy, firstly due to

the ease by which assertions can be checked, and secondly because of the similarity between the Alloy Language and Z. We then introduced the main features of Alloy. We demonstrated how our framework can be represented in Alloy. We showed how assertions can be defined and used to check the consistency of scenarios to policy definitions. Finally, we demonstrated how Alloy models can be divided into modules. This breakdown into modules aids scalability by reducing the size of the model required for each scenario that is to be analysed.

# Chapter 8

# Case Study: A Bank

We have already used two case studies in this thesis, the first one to introduce the policy framework in chapter 5, and the second one in chapters 4 and 6 to illustrate how formal Tropos could be extended. The case study presented in this chapter demonstrates how a formal Tropos policy model can be constructed, translated into Alloy, and analysed using the Alloy tool. The selected case study from the literature (Schaad, 2003) explores several principles of management control, including the minimum privileges principle, delegation, and the separation of duties, making it particularly well suited to exploring access policies. Here we continue to focus on the minimum privileges principle.

## 8.1 Case Study Description

The case study is based on an access control system of a European bank. The bank has 50,000 employees, over a thousand branches, and provides banking services for local communities. Schaad (2003) reviews the bank's access control system, and how it satisfies organisational control principles. Although the focus is on the access control system, many of the requirements can be inferred from it. We consider the requirements of a system for a branch, and consider a few requirements identified by Schaad.

One of the key services is that of providing credit, for example, extending an overdraft, providing a mortgage, or offering a sum of money. Each of these involves different actors, and different information assets. The controls to be applied to these services also differ. We focus on the requirements of one of these services: the provision of a sum of money. The flow diagram in figure 8.1 shows some of the steps involved.

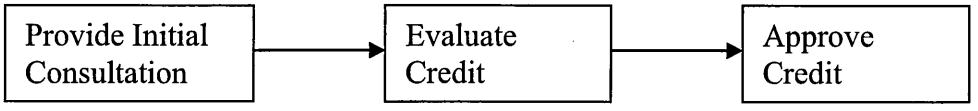



**Figure 8.1 Credit Application Process**

This service is carried out by the group, customer advisory services. The provision of an initial consultation and the evaluation of credit are carried out by the customer advisory clerks. The approval of credit is done by the advisor's manager. The function customer advisory services is carried out within a branch; within each branch are several hierarchies of authority, for each of the different specialised functions. The head of a branch is responsible for general banking services, and has a personnel function, dealing with disciplinary matters for example, but the management of specialised functions, such as customer advisory services, is achieved through its own hierarchy; thus a customer advisor clerk would take instructions from a manager in the same function to whom he is assigned rather than from the branch manager. Another function within a branch is share trading; there is a strict separation of duties between the functions customer advisory services and share trading within a branch.

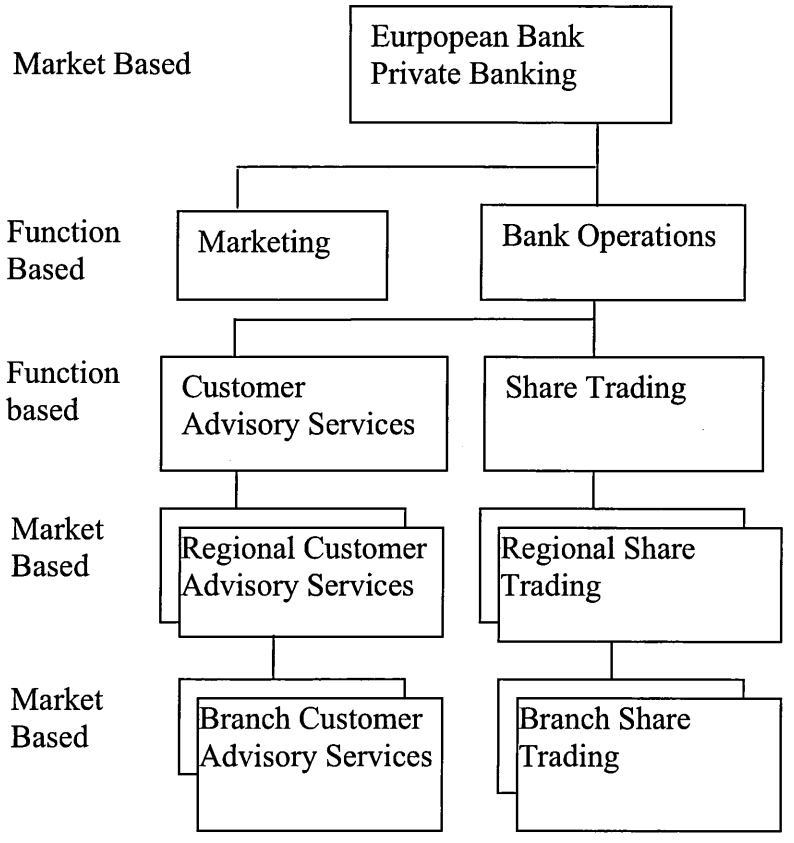



**Figure 8.2 Organisational Structure of a European Bank**

## 8.2 Deriving the Policy Model

In deriving actor definitions for our policies, the first step is to define the groupings within the organisation. The groupings form a composite structure. For the bank this is represented in figure 8.2.

We can then identify whether a grouping represents a domain in that it exists to serve a specific market or whether it is purely functional. From these groupings we can then derive the organisational functions and domains that are as follows:

**Organisational Function** Customer Advisory Services

**Organisational Function** Share Trading

**Organisational Domain** Region

**Organisational Domain** Branch
  **Part** Region

Within each grouping there is a hierarchical structure. Focusing on the function, customer advisory services, in a branch, there exist the following levels of authority. In decreasing order of authority they are:

**Authority** Head of Branch

**Authority** Manager
  **Senior** Head of Branch

**Authority** Clerk.
  **Senior** Manager

The definition of seniority levels is necessary to distinguish roles within the same domain and organisational function. For defining the minimum privileges it is not necessary to know which role is senior, nevertheless, if we were to define delegation policies, then it becomes useful.

We can now define positions within these groups, where an actor definition is created for each level of authority. For example, the following definition shows the Customer Advisory Services Manager position associated with the organisational function Customer Advisory Services:

**Actor** Customer Advisory Services Manager
  **Type** Role
  **Organisational Function** Customer Advisory Services
  **Organisational Domain** Branch
  **Authority** Manager

A similar definition can be given for a Clerk. We can now define the tasks and the resources associated with these tasks:

**Task** Initial Consultation
  **Resource** Credit Application

**Task** Evaluate Credit
  **Resource** Credit Application
  **Resource** Credit History

**Task** Approve Credit
  **Resource** Credit Application

This enables us to extend our actor definitions with task assignments and hence create policies. The minimum privileges policies associated with the Customer Advisory Services Manager and Clerk are:

**Actor** Customer Advisory Services Manager
  **Type** Role
  **Organisational Function** Customer Advisory Services
  **Organisational Domain** Branch
  **Authority** Manager
  **Task** Approve Credit

**Actor** Customer Advisory Services Clerk
  **Type** Role
  **Organisational Function** Customer Advisory Services
  **Organisational Domain** Branch
  **Authority** Clerk
  **Task** Initial Consultation
  **Task** Evaluate Credit

The authority levels of Manager and Clerk are applicable to different functional groupings. For example, there are clerks assigned to Customer Advisory Services, other clerks assigned to Share Trading, and so on. A manager is distinguished from a clerk in that he has the authority to delegate tasks to clerks. In order for a clerk or manager to be able to execute a function, they need to be assigned to a functional grouping in a specific branch. Hence the actor definition is a composition of the level of authority, organisational function, and organisational domain.

We can now demonstrate how the formal Tropos definitions map onto Alloy using the rules that we defined in the previous chapter. The authority level of Manager translates into the following Alloy construct using the corresponding meta-concept translation rule for authority levels:

```
one sig manager extends Authority {}
```

The translation of the authority level of Clerk and the seniority relationship to a Manager maps to Alloy using the authority hierarchy translation rule as follows:

```
one sig clerk extends Authority {}
fact { clerk.senior = manager }
```

Similarly, mappings are carried out for organisational functions, organisational domains, tasks, and resources, which we will not repeat here. The following role definition is mapped from the actor definition of a Customer Advisory Services Manager using the corresponding meta-concept translation rule for an actor:

```
one sig customer_advisory_services_manager extends Role {}
fact { customer_advisory_services_manager.org_domain = branch }
fact { customer_advisory_services_manager.authority = manager }
fact { customer_advisory_services_manager.org_function =
customer_advisory_services }
```

The extended actor definition for the Customer Advisory Services Manager, which includes the task assignment Approve Credit represents a restriction that translates into Alloy using the policy definition translation rule as follows:

```
one sig approve_credit_policy extends Authorisation_Policy {}
fact { approve_credit_policy.task = approve_credit }
fact { approve_credit_policy.role = customer_advisory_services_manager }
```

The prerequisite for this definition is that the role and task definitions already exist. Similarly. For the other tasks such as Initial Consultation and Evaluate Credit, we can also define corresponding policies.

The next step is to define an instantiation to verify the policy. In the following instantiation, we check that a Customer Advisory Services Manager can approve the credit of a customer of the branch to which he is assigned. First, we define two domain instantiations for the Frankfurt and Dortmund branches:

**Organisational Domain** Frankfurt Branch **INS** Branch

**Organisational Domain** Dortmund Branch **INS** Branch

Then, we can define an instantiation of a Customer Advisory Services Manager in the Frankfurt Branch:

**Actor** Customer Advisory Services Manager Frankfurt **INS** Customer Advisory Services Manager
  **Type** Role
  **Organisational Domain** Frankfurt Branch

These two Tropos definitions are mapped into Alloy using the instantiation translation rules. The following is a definition of the Frankfurt Branch that instantiates Branch; i.e. it is a branch:

```
one sig frankfurt_branch extends Org_Domain {}
fact { frankfurt_branch.insd = branch }
```

The following definition represents the instantiated role for a Customer Advisory Services Manager in the Frankfurt Branch:

```
one sig customer_advisory_services_manager_frankfurt
fact { customer_advisory_services_manager_frankfurt.insr =
customer_advisory_services_manager }
fact { customer_advisory_services_manager_frankfurt.org_domain = frankfurt }
```

We also need to define the instantiations of assets and tasks. We first define the assets Credit Application and Credit History of the customer Philip Stokes. We assign these assets to the Frankfurt Branch:

**Resource** Credit Application of Philip Stokes **INS** Credit Application
  **Organisational Domain** Frankfurt Branch

**Resource** Credit History of Philip Stokes **INS** Credit History
  **Organisational Domain** Frankfurt Branch

We then define instantiations of the tasks Approve Credit Application and Initial Consultation for the credit application of the customer Philip Stokes:

**Task** Approve Credit Application of Philip Stokes **INS** Approve Credit Application
  **Resource** Credit Application of Philip Stokes

**Task** Initial Consultation for Philip Stokes **INS** Initial Consultation
  **Resource** Credit Application of Philip Stokes
  **Resource** Credit History of Philip Stokes

The definitions for the task Approve Credit Application of Philip Stoke and the resource Credit Application of Philip Stokes translate into our policy framework as follows

```
one sig philip_stokes_credit_application extends Asset {}
fact { philip_stokes_credit_application.insa = credit_application }
fact { philip_stokes_credit_application.asset_domain = frankfurt_branch }

one sig approve_credit_application_of_philip_stokes extends Task {}
fact { approve_credit_application_of_philip_stokes.inst = approve_credit_application }
fact { approve_credit_application_of_philip_stokes.task_asset_dependency =
credit_application_of_philip_stokes }
```

The translation of the resource Credit History of Philip Stokes and the task Initial Consultation for Philip Stokes is similar.

Finally we define the scenario of Jim Smith occupying the role of the Customer Advisory Services Manager Frankfurt executing the task Approve Credit Application of Philip Stokes:

**Actor** Jim Smith **OCCUPIES** Customer Advisory Services Manager Frankfurt
  **Type** Agent
  **Task** Approve Credit Application of Philip Stokes

This agent definition is mapped into Alloy as follows:

```
one sig jim_smith extends Agent {}
fact { jim_smith.role_assignment = customer_advisory_services_manager_frankfurt }
fact { jim_smith.performs = approve_credit_application_of_philip_stokes }
```

We can now check the model by defining assertions. When a check command is executed, Alloy searches for a counter example which breaks the assertion, and then will display the state by which the solution is arrived at, otherwise the tool simply states that no solution was found. For the purposes of demonstrating an Alloy check, a false assertion is therefore more informative. We can demonstrate this with the following assertion that Jim Smith who is a manager can not approve a credit application.

```
assert execute_approve_credit_jim_smith {
all task: approve_credit_application_of_philip_stokes, ag: jim_smith |
task !in ag.performs }
```

This can be checked by using the following check statement:

```
check execute_approve_credit_jim_smith for 4 but 2 Org_Function, 5 Task,
3 Authority, 3 Authorisation_Policy, 1 Agent
```

This check statement includes the number of instances that Alloy should generate for each signature type. For example we have defined the two instances, share_trading and customer_advisory_services, of the signature Org_Function; we therefore limit Alloy to generating these two organisational functions. If we were to define more than two, Alloy would generate additional instances itself; if we were to define less, then Alloy would produce an error. A default of four is given, so that Alloy will generate four instances of any signature type for which an explicit number of instances has not been given.

This assertion is a negation of what is required, and therefore we expect Alloy to find a solution. This is shown in figure 8.3.

We can now a similar assertion this time though to test whether Jim Smith can carry out an initial consultation on the credit application. In formal Tropos this scenario is as follows:

**Actor** Jim Smith **OCCUPIES** Customer Advisory Services Manager Frankfurt
  **Type** Agent
  **Task** Initial Consultation for Philip Stokes

In Alloy we define this as follows:

```
one sig jim_smith extends Agent {}
fact { jim_smith.role_assignment = customer_advisory_services_manager_frankfurt }
fact { jim_smith.performs = initial_consulation_for_philip_stokes }
```

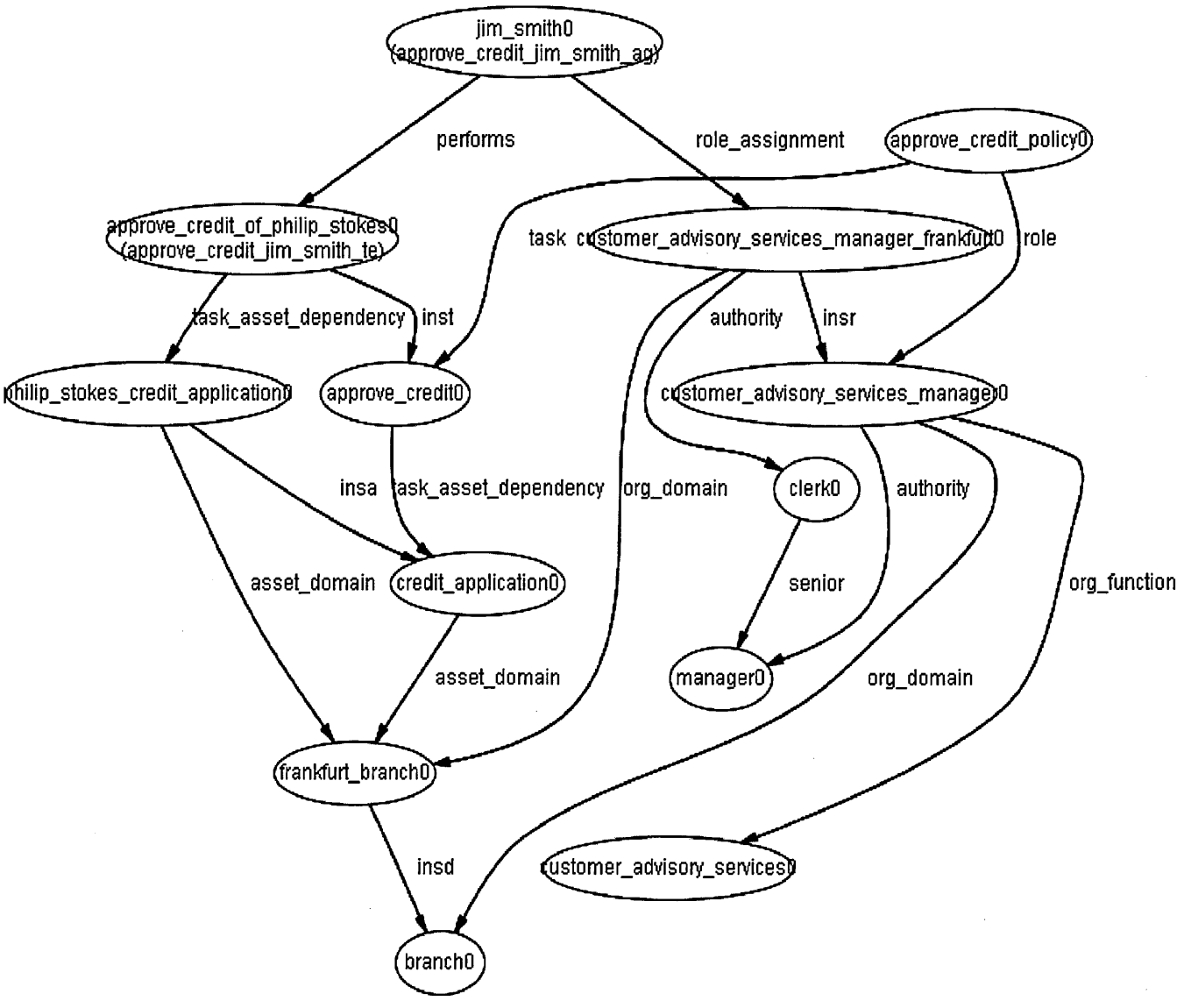



**Figure 8.3 Task Execution Scenario in Alloy of Approve Credit Application**

The assertion and check definitions in Alloy are as follows:

```
assert execute_initial_consultation_jim_smith {
all task: initial_consultation_for_philip_stokes, ag: jim_smith | !task in ag.performs }

check execute_initial_consultation_jim_smith for 4 but 2 Org_Function, 5 Task,
3 Authority, 3 Authorisation_Policy, 1 Agent
```

This time the tool does not find an example, demonstrating that Jim Smith can not actually carry out an initial consultation. This is consistent with the policy defined above, that only allows clerks can perform initial consultations.

## 8.3 Model Consistency Checks

The framework that we introduced in chapter 5 includes invariants that need to be maintained to ensure consistency. In chapter 7 we then translated these invariants into Alloy. For example an authority level can not be senior to itself, similarly constraints exist to prevent circular definitions in other hierarchies, and there are constraints to prevent instantiations being included in policy domain definitions. In this section we demonstrate, using the case study, a couple of examples of how definitions that violate these constraints can be identified using Alloy assertions.

The first example concerns invalid authority definitions and is as follows:

**Authority** Clerk
  **Senior** Manager

**Authority** Manager
  **Senior** Clerk

Here we have defined the authority level of Clerk and Manager that are both senior to one another. These definitions violate the framework constraint that a level of authority can not be senior to itself. This can be checked by including the constraint in an assertion and running a check, as follows:

```
assert authority_level_consistent {
all au: Authority | au !in au.^senior }

check authority_level_consistent for 4 but 2 Org_Function, 5 Task, 3 Authority,
3 Authorisation_Policy, 1 Agent
```

In this case Alloy finds a solution indicating that the constraint for authority levels has been violated.

The second example concerns an invalid role instantiation definition and is as follows:

**Actor** Customer Advisory Services Manager Dortmund **INS** Customer Advisory Services Manager Frankfurt
**Type** Role
**Organisational Domain** Dortmund Branch

Here we have defined a role that is instantiated from a role that is itself an instantiated role. This violates a framework meta-model constraint. This can be checked using the following assertion and running a check:

```
assert instantiated_role_consistent {
all role1, role2: Role | role1.insr = role2 =>
role2.insr = none }

check instantiated_role_consistent for 4 but 2 Org_Function, 5 Task, 5 Role,
3 Authority, 3 Authorisation_Policy, 1 Agent
```

In this case Alloy finds a solution indicating that the constraint for role instantiation has been violated.

## 8.4 Evaluation of Extended Formal Tropos

For this case study it is worth reflecting on how this extended formal Tropos improves on existing approaches.

First of all the heuristics that we proposed enabled us to derive the actor definitions systematically, by deriving them from the organisational structure. The approach that we have adopted begins by defining organisational functions, domains and authority levels within these domains and hence to construct actor definitions from these, as we have demonstrated. In this way we derived two abstract actor definitions, customer advisory services clerk, and customer advisory services manager, from which we could instantiate into the respective actors in a specific branch. The current Tropos approach does not include such a set of heuristics. One approach that does have a set of heuristics, the ReCAPS role engineering approach proposed by He (2005) that we reviewed in chapter 2, derives role definitions from collections of tasks. However, although they can define the functional characteristics,

the seniority and organisational domains are missing. In this particular case study the organisational domain, the branch, is vital in order to define policies based on the minimum privileges principle. The task, approve credit, that we have modelled is actually a step required to satisfy the principle of supervision and review, and the seniority relationship that we have defined is necessary to differentiate between different levels in the organisation; i.e. that the manager is supervising the clerk. The model that we have defined also makes clear that this supervisory relationship applies only within a branch and within the organisational function, customer advisory services. These characteristics can not be modelled in Tropos as it currently is.

We reviewed two approaches which do include the organisational context in chapter 2. ORDIT (Dobson *et al.*, 1992) does have a role model which captures the hierarchical relationships between roles; as we described in chapter 2 in ORDIT power relationships between roles can be modelled and so enabling us to some extent to model the delegation of obligations. However a key element is missing is the organisational domain which means that relating a role to a branch would be not be a part of the model. Our model has also been defined formally allowing us reason about it and in particular verify that scenarios, such as we defined in the case study, are consistent with the policies defined.

## 8.5 Chapter Summary

In this chapter we presented a case study of how the framework in extended formal Tropos we presented in chapter 6, can be applied. The case study was taken from the literature of a the access policies of a large European bank. We first created a model of the organisational context, from which we then derived role definitions. We identified a few tasks carried out by the customer advisory services in a branch and

assets which require access. We then defined policies to satisfy the minimum privileges principle with respect to these tasks and scenarios to verify the policies. We demonstrated how the formal Tropos model can be translated into the Alloy language and how one of the policies could be verified using an assertion. Finally we outlined the key advantages of this approach compared to existing alternatives, in that the inclusion of the organisation context, can enable us to define more precise policies based on the principle of minimum privileges principle and also enables us to define hierarchical relationships that provide a basis for the definition of principles based on delegation, and supervision and review.

# Chapter 9

# Discussion and Conclusions

In this chapter we first present a summary of the thesis, then an analysis of the contributions, and finally a discussion and critical evaluation of the research presented in the thesis, which gives an account of future work.

## 9.1 Thesis Summary

In this thesis we have addressed the problem of modelling access policies to ensure that security goals can be achieved, and that operational requirements are consistent with access policies.

We first identified the importance of an organisational analysis before making actor or role definitions in the context of modelling access policies. We highlighted the lack of this in current modelling approaches, thus making it difficult to express access policies precisely, and also to refine them into operational constraints.

We proposed a framework that comprises a meta-model for formally modelling the organisational context, and deriving organisational role definitions. It also includes a set of heuristics as to how to identify groupings, the levels of authority and management domains, from which roles can be defined. We defined the meta-model in Z so enabling us to reason about it, and we demonstrated how automated checking could be carried out through translation into the specification language Alloy.

We showed how this framework could be integrated into Tropos, an extended version of the i* framework, illustrating the complementary nature of the new framework to at least one existing requirements modelling approach.

## 9.2 Analysis of Contributions

The thesis makes a contribution to the modelling of access policies as requirements. In chapter 2 we identified a key weakness with respect to modelling policies that are derived from the principles of management control, in that actors are not linked to the organisational context. This makes it difficult to define policies to satisfy the minimum privileges principle. In chapter 3 we saw that policy languages such as Ponder can define these policies, because they are based on mapping groups and roles onto the organisational context, exemplified in Ponder by mapping groups onto organisational domains, and defining authority as management structures.

The framework we presented addresses the need to define and verify access policies in requirements models, rather than only being able to do this effectively at the implementation level. In effect it describes an enriched ontology based on concepts that we identified in the organisational literature. It is derived from the two key dimensions on which organisations are structured that we identified in chapter 3: the division of work, and the lines of authority. The framework meta-model includes meta-concepts that enable us to model these characteristics. We also outlined a set of heuristics, which give us a systematic approach to deriving these organisational characteristics; determined from the organisational structure on the basis by which groups are structured, either on a functional or market basis. This process ensures that roles are linked to the organisational context, with the advantage of enabling a more precise definition of what a role is.

We hypothesised in chapter 2, that linking the role to the organisational context is a prerequisite for defining policies that satisfy the key principles of management control. In this thesis we focused on the minimum privileges principle, and the framework includes a construct for defining access policies based on this principle that assign tasks to roles, with tasks being related to the assets that are required to perform the task. The addition of organisational domains into the role definition, and the relationship between the role and the organisational domain to which assets are assigned, allows us to define policies that fully satisfy the minimum privileges principle. The contrast between policy constructs in our framework and policy frameworks used for access control, is that in our framework the policies are abstract, whereas policies defined for access control systems are based on instances.

We also addressed the need to be able to verify policies. As we identified in chapter 2, scenarios are an effective way of verifying requirements. The framework includes meta-concepts and constructs that enable us to define instantiations of organisational groupings, roles and agents that are assigned to role instantiations. This enables us to generate scenarios and then verify that the policies satisfy the minimum privileges principle. The fact that we defined the framework formally in Z gives a basis for reasoning about the consistency between policies and scenarios. As we outlined in chapter 7, performing proofs in Z, even with the support of tool, is an arduous process; that is the reason why we used the modelling tool Alloy for this purpose. We demonstrated how the Z constructs can be translated into the Alloy language, and how automated analysis can be carried out using the tool. The fact that a large model can be divided into modules, means that we can scale the approach to analysing systems with a large number of roles by separating the modules to map on to different parts of the organisation, and to separate policies and scenarios. The point

of this demonstration is that we showed how a tool could be constructed based on the framework for defining policies and verifying scenarios.

A key objective, we outlined in this thesis, was to relate the framework to an existing requirements modelling language. For investigation we selected the i* framework and formal Tropos. We demonstrated how the organisational meta-model presented in chapter 5 can be applied to extend formal Tropos. We then examined how the extended formal Tropos language can be applied to define access policies, and scenarios. We showed how the access policies defined in formal Tropos can be translated into Alloy.

We used a case study from the literature to demonstrate how an extended version of formal Tropos could be used to define access policies. The case study concerned a large European bank, and we showed how policies satisfying the minimum privileges principle could be defined, based on a derivation of the organisational context using the heuristics we presented in chapter 5. We also demonstrated how policies and scenarios can be translated into the Alloy language and analysed using the tool.

In chapter 2 we summarised the capabilities of requirements modelling approaches with respect to management control principles. In table 2.2 we highlighted that none of the principles of management control could be adequately defined. Table 9.1 overleaf highlights the key contribution of our work, namely that the extended Tropos presented in this thesis enables us to define the minimum privileges principle.

In chapter 2 we also presented, in table 2.3, the capabilities of requirements modelling approaches with respect to modelling the organisational context. In table 9.2, we highlight that extended Tropos now allows us to model organisational domains, organisational functions, and authority relationships. It is these definitions that are required to define policies that satisfy the minimum privileges principle.

However we also have a basis for defining other principles including delegation, and supervision and review.

| Management Control Principle | Extended Tropos |
|---|---|
| Minimum Privileges Principle | yes |
| Segregation of Duties | partially |
| Delegation and Revocation of Authority | partially |
| Supervision and Review | no |
| Accounting Principles | no |

**Table 9.1 Coverage of Management Control Principles by Extended Tropos**

| Modelling of the Organisational Context | Extended Tropos |
|---|---|
| Agent Assignments to Tasks and Resources | yes |
| Separation of Roles to Agents | yes |
| Organisational Domains | yes |
| Organisational Functions | yes |
| Authority Relationships | yes |
| Workflow | no |

**Table 9.2 Modelling of the Organisational Context by Extended Tropos**

## 9.3 Critical Analysis and Future Work

In chapter 2 we identified several commonly used management control principles. In this thesis we have only explored the minimum privileges principle. Other principles remain to be explored. Although the framework includes authority relationships, we have not demonstrated how we can define policies that satisfy the delegation and revocation of authority, or supervision and review. An opportunity for further research would be to extend the approach proposed by Giorgini *et al.* (2005) for modelling delegation using the i* framework. In particular, accounting principles, can lead to complex procedures, whereby workflows need to be modelled and financial constraints such as credit ratings need to be included in policies.

In defining the organisational context the examples we explored were role cultures, typical of large organisations. In fact the identification of roles and the link to the organisational is likely to be much easier in a large organisation such as a bank or hospital. Although these types of organisations figure prominently in the security literature many organisations particularly small organisations are much less formal in their structures. A research question therefore is to what extent this framework is valid for other organisational cultures and could it be adapted or extended? Furthermore, the organisational modelling approach that we have proposed we applied to single organisations, however systems can be integrated across organisational boundaries, a further question is therefore is to what extent would the framework enable us to model this type of organisational context?

We have demonstrated how to extend formal Tropos to define policies, but there are other approaches to modelling. Defining use cases is a widely used approach to modelling requirements; it would therefore be useful to explore how use cases could

be extended, using the framework. Our demonstration of how actor definitions can be extended in formal Tropos could be used as a basis for this. Indeed the principle of how to define actors or agents and link them to the organisational context ought to be able to be applied to any modelling approach, by extending the syntax.

Although we have demonstrated that it is possible to analyse policies in the Alloy tool, if this were to become an industrial approach, then a tool would be desirable that would enable policy definitions to be defined in a requirements modelling language rather than Alloy, for example formal Tropos, or perhaps use cases. The formal Tropos approach uses NuSMV, which enables temporal constraints to be modelled, which could potentially be useful for modelling policies based on accounting principles. Thus a potential avenue for research would be to investigate how to translate extended formal Tropos definitions into NuSMV. Use cases are not generally formally defined, and hence if the automated checking were to be carried out a formalisation of use cases would be required.

The validation of our approach was based on a limited set of hypothetical case studies. In order to determine the extent to which this approach would work in practice, it is necessary to actually carry out projects. It is only by practical experience that an approach can be improved and refined.

## 9.4 Conclusions

The main objective of this thesis was to address the problem of defining access policies and refining them into constraints. We identified the nature of access policies, and also the principles by which organisations are structured and controlled. These principles are fundamental to the understanding of the requirements of access policies. The framework we have proposed includes the macro-organisational context, which

makes it much easier to derive precise role definitions. This is one significant weakness of existing modelling approaches.

We have focused on the principle of minimum privileges, however in doing this we have created a foundation for other principles. For example the framework relates authority levels, which give us a basis for defining delegation. The definitions of organisational domain and function provide a basis for defining the segregation of duties.

The formal notation that we adopted in developing the framework enables us to reason about it. We demonstrated this using the Alloy tool, which enables assertions to be checked automatically.

The motivation for the research, was that in current modelling approaches there is a weak link between actors defined in a requirements model and the organisational context. In this thesis we have demonstrated that the framework we presented strengthens that link. It is complementary to other modelling approaches, in that it only focuses on the link to the organisational context, but does not prescribe how other aspects of a requirements model such as goals, functions, tasks and resources, should be defined. The framework can thus be used to further develop other requirements modelling approaches as we demonstrated with the i* framework.

# Bibliography

Abdallah, A., and Khayat, E. J. "A Formal Model for Parameterized Role-Based Access Control," Formal Aspects in Security and Trust: 2nd IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), Toulouse, France, 22-27th August, 2004, pp. 233-246.

Alspaugh, T. A., Faulk, S. R., Britton, K. H., Parker, R. A., Parnas, D. L., and Shore, J. E. "Software Requirements for the A-7E Aircraft," NRL Memorandum Report 3876, Naval Research Lab., Washington, DC, USA, 1992.

Alexander, I. "Modelling the Interplay of Conflicting Goals with Use and Misuse Cases," 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02), Essen, Germany, 9-10th September, 2002, pp. 145-152.

Anderson, R. "A Security Policy Model for Clinical Information Systems," IEEE Symposium on Security and Privacy, Oakland, California, USA, 6-8th May, 1996, pp. 30-45.

Anderson, R. *Security Engineering - A Guide to building dependable distributed Systems*, John Wiley & Sons Inc, 2001.

Antón, A.I. "Goal-Based Requirements Analysis," 2nd IEEE International Conference on Requirements Engineering (ICRE'96), Colorado Springs, Colorado, USA, 15-18th April, 1996, pp. 136-144.

Antón, A.I. "Goal Identification and Refinement in the Specification of Software-Based Information Systems," PhD Thesis, Georgia Institute of Technology, Atlanta, Georgia, USA, 1997.

Antón, A.I., and Earp, J.B. "Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems," *Recent Advances in E-Commerce Security and Privacy*, Gosh, A.K. (ed.), Kluwer Academic Publishers, 2001, pp. 29-46.

Antón, A.I., Earp, J.B., Potts, C., and Alspaugh, T.A. "The Role of Policy and Stakeholder Privacy Values in Requirements Engineering," IEEE 5th International Symposium on Requirements Engineering (RE'01), Toronto, Canada, 27-31st August, 2001, pp. 138-145.

Bacon, J., Lloyd, M., and Moody, K. "Translating Role-Based Access Control within Context," 2nd IEEE International Workshop on Policies for Distributed Systems and Networks (Policy 2001), Bristol, UK, 29-31st January, 2001, pp. 107-119.

Bandara, A.K., Lupu, E.C., Moffett, J., and Russo, A. "A Goal-based Approach to Policy Refinement," 5th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy 2004), New York, USA, 7-9th June, 2004, pp. 229-239.

Barka, E., and Sandhu, R. "A Framework for Role Based Delegation Models," 16th Annual Computer Security Applications Conference (ASAC'00), New Orleans, Louisiana, USA, 11-15th December, 2000, pp. 168-176.

Bell, D., and LaPadula, L.J. "Secure Computer Systems: A Mathematical Model," *MITRE Technical Report 2547* (2), 1973.

Benner, K.M., Feather, M.S., Johnson, W.L., and Zorman, L.A. "Utilizing Scenarios in the Software Development Process," *Information System Development Process*, 1993, pp. 117-134.

Bertino, E., Bonatti, P.A., and Ferrari, E. "TRBAC: Temporal Role-Based Access Control Model," 5th ACM Workshop on Role-Based Access Control, Berlin, Germany, 26-27th July, 2000, pp. 21-30.

Biddle, B.J. *Role Theory: Expectations, Identities and Behaviours*, Academic Press, 1979.

Breaux, T.D., and Antón, A.I "Analyzing Goal Semantics for Rights Permissions and Obligations," IEEE International Requirements Engineering Conference (RE05), Paris, France, 29th August -1st September, 2005, pp. 177-186.

Brown, S.J., and Steenbeek, O.W. "Doubling: Nick Leeson's Strategy," *Pacific Basin Journal* (9:2), April 2001, pp. 83-99.

Büchi, M. "The B Bank: A Complete Case Study," 2nd International Conference on Formal Engineering Methods (ICFEM'98), Brisbane, Australia, 9-11th December, 1998, pp. 190-199.

Buchanan, D.A., and Huczynski, A.A. *Organizational Behaviour: An Introductory Text*, Prentice Hall International, 1985.

Carnot, M., DaSilva, C., Dehbonei, B., and Mejia, F. "Error-free Software Development for Critical Systems using the B-Methodology," 3rd International Symposium on Software Reliability Engineering, Research Triangle Park, North Carolina, USA, 7-10th October, 1992, pp. 274 - 281.

Chung, L. "Representation and Utilization of Non-Functional Requirements for Information System Design," Advanced Information Systems Engineering 3rd International Conference (CaiSE'91), Trondheim, Norway, 13-15th May, 1991, pp. 5-30.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. "NUSMV 2 An Open Source Tool for Model Checking," Computer Aided Verification Conference, number 2404 in Lecture Notes in Computer Science, Springer, Copenhagen, Denmark, 27-31st July, 2002, pp. 241-268.

Clark, D.D., and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies," IEEE Symposium on Security and Privacy, 1987, pp. 184-194.

Clarke, E., and Wing, J. "Formal Methods: State of the Art and Future Directions," *ACM Computing Surveys* (28:4), 1996, pp. 626-643.

Cockburn, A. *Writing Effective Use Cases,* Addison Wesley, 2001.

Covington, M.J., Long, W., Srinivasan, S., Dey, A.K., Ahamad, M., and Abowd, G.D. "Securing Context-Aware Applications Using Environment Roles," 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, 3-4th May, 2001, pp. 10-20.

Coyne, E.J. "Role Engineering," 1st ACM Workshop on Role-Based Access Control, Gaithersburg, Maryland, USA, 30th November -1st December, 1995, pp. 115-116.

Crook, R., Ince, D., and Nuseibeh, B. "Towards an Analytical Role Modelling Framework," 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02), Essen, Germany, 9-10th September, 2002a, pp. 123-136.

Crook, R., Ince, D., and Nuseibeh, B. "Modelling Access Policies using Roles in Requirements Engineering," *Information and Software Technology* (45:14), November 2003, pp. 971-991.

Crook, R., Ince, D., and Nuseibeh, B. "On Modelling Access Policies: Relating Roles to the Organisational Context," IEEE International Requirements Engineering Conference (RE'05), Paris, France, 29th August -1st September, 2005, pp. 157-166.

Crook, R., Nuseibeh, B., Lin, L., and Ince, D. "Security Requirements Engineering: When Anti-Requirements Hit the Fan," IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 11-13th September, 2002b, pp. 203-205.

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "Ponder: A Language for Specifying Management and Security Policies for Distributed Systems," Research Report, Imperial College, London, UK, 2000.

Dardenne, A., van Lamsweerde, A., and Fickas, S. "Goal-Directed Requirements Acquisition," *Science of Computer Programming* (20:1-2), 1993, pp. 3-50.

Dobson, J.E., Blyth, A.J.C., Chudge, J., and Strens, M.R. "The ORDIT Approach to Requirements Identification," 16th Annual International Computer Software and Applications Conference (COMPSAC '92), Chicago, Illinois, USA, 21-25th September, 1992, pp. 356 -361.

Dobson, J.E., and Strens, M.R. "Organisational Requirements Definition for Information Technology Systems," 1st IEEE International Conference on Requirements Engineering (ICRE'94), Colorado Springs, Colorado, USA, 18-22nd April, 1994, pp. 158-165.

Fernandez, E.B., and Hawkins, J.C. "Determining Role Rights from Use Cases," 2nd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 6-7th November 1997, pp. 121-125.

Fitzgerald, J., and Larsen, G.L., *Modelling Systems: Practical Tools and Techniques in Software Development*, Cambridge University Press, 1998.

Fontaine, J.P. "Goal Oriented Elaboration of Security Requirements," Masters Dissertation, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 2001.

Fuxman, A., Pistore, M., Mylopoulos, J., and Traverso, P. "Model Checking Early Requirements Specifications in Tropos," 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, 27-31st August, 2001, pp. 174-181.

Georgiadis, C.K., Mavridis, I., Pangalos, G., and Thomas, R.K. "Flexible Team-Based Access Control Using Contexts," 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, 2001, pp. 21-27.

Giorgini, P., Kolp, M.,Mylopoulos, J., and Pistore, M. "The Tropos Methodology: an Overview," *Methodologies and Software Engineering for Agent Systems*, Bergenti, F.,Gleizes, M.-P., and Zambonelli, F (ed.), Kluwer Academic Publishing, 2004.

Giorgini, P., Massacci, F., Mylopolous, J. and Zannone, N. "Modeling Security Requirements Through Ownership Permission and Delegation," 13th IEEE International Requirements Engineering Conference (RE'05), Paris, France, 29th August -1st September, 2005, pp. 167 -176.

Glinz, M. "An Integrated Formal Model of Scenarios Based on Statecharts," 5th European Software Engineering Conference (ESEC'95), Sitges, Spain, 25-28th September, 1995, pp. 254-271.

Haley, C.B., Laney, R., Moffett, J.D., and Nuseibeh, B. "The Effect of Trust Assumptions on the Elaboration of Security Requirements," 12th International Requirements Engineering Conference (RE'04), Kyoto, Japan, 6-10th September, 2004, pp. 102-111.

Haley, C.B., Moffett, J.D., Laney, R., and Nuseibeh, B. "Arguing Security: Validating Security Requirements Using Structured Argumentation," 3rd Symposium on Requirements Engineering for Information Security (SREIS'05), Paris, France, 29th August, 2005.

Handy, C. *Understanding Organisations*, Penguin, 1985.

He, Q., and Antón, A. "A Framework for Modeling Privacy Requirements in Role Engineering," 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03), Klagenfurt/Velden, Austria, 16-17th June, 2003, pp. 117-124.

He, Q. "Requirements-Based Access Control Analysis and Policy Specification," PhD Thesis, North Carolina State University, Raleigh, North Carolina, USA, 2005.

He, Q., Otto, P., Antón, A., and Jones, L. "Ensuring Compliance between Policies, Requirements and Software Design: A Case Study," 4th IEEE International Workshop on Information Assurance (IWIA'06), Raleigh, North Carolina, USA, 13-14th April, 2006, pp. 79-92.

Heitmeyer, C.L. "Using the SCR Toolset to Specify Software Requirements," 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT'98), Boca Raton, Florida, USA, 19th October, 1998, pp. 12-14.

Heitmeyer, C.L. "Software Cost Reduction," *Encyclopedia of Software Engineering*, Marciniak, J.J. (ed.), John Wiley and Sons, 2002, pp. 1374-1380.

Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y., and Chen, C. "Formal Approach to Scenario Analysis," *IEEE Software* (11:2), March 1994, pp. 33-41.

ISO "ISO/IEC 17799:2005 Information Technology – Security Techniques – Code of Practice for Information Security Management," International Organization for Standardization, 2005.

Jackson, D. "Micromodels of Software: Lightweight Modelling and Analysis with Alloy Software, Design Group. MIT Lab. for Computer Science," 2004

Jackson, M. *Problem Frames Analyzing and Structuring Software Development Problems*, Addison-Wesley, 2001.

Jackson, M., and Zave, P. "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology* (6:1), 1997, pp. 1-30.

Jajodia, S., Samarati, P., and Sabrahmanian, V.S. "A Logical Language for Expressing Authorisations," IEEE Symposium on Security and Privacy, Oakland, California, USA, 4-7th May, 1997, pp. 31-42.

Jarke, M., Bubenko, J., Rolland, C., Sutcliffe, A.G., and Vassilou, Y. "Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis," 1st IEEE Symposium on Requirements Engineering, San Diego, California, USA, 4-6th January, 1993, pp. 19-31.

Jones, C.B. *Systematic Software Development Using VDM*, (2nd Edition), Prentice Hall, 1990.

Lin, L., Nuseibeh, B., Ince, D., Jackson, M., and Moffett, J. "Introducing Abuse Frames for Analysing Security Requirements," 11th IEEE International Requirements Engineering Conference (RE'03), Monterey, California, USA, 8-12th September, 2003, pp. 371-372.

Lin, L., Nuseibeh, B., and Ince, D. "Using Abuse Frames to Bound the Scope of Security Problems," 3rd International Workshop on Requirements for High Assurance Systems (RHAS'04), Kyoto, Japan, 6th September, 2004, pp. 29-34.

Liu, L., Yu, E., and Mylopolous, J. "Security and Privacy Requirements Analysis within a Social Setting," IEEE international Conference on Requirements Engineering (RE'03), Monterey, California, USA, 8-12th September, 2003, pp. 151-161.

Loucopoulos, P., and Kavakli, E. "Enterprise Modelling and the Teological Approach to Requirements Engineering," *International Journal of Cooperative Information Systems* (4:1), 1995, pp. 45-79.

Lupu, E., Sloman, M., Dulay, N., and Damianou, N. "Ponder: Realising Enterprise Viewpoint Concepts," 4th International Conference on Distributed Object Computing, Mukahari, Japan, 2000, pp. 66-75.

Maiden, N.A.M. "CREWS-SAVRE Scenarios for Acquiring and Validating Requirements," *Journal of Automated Software Engineering* (5), 1998, pp. 419-446.

Massaci F., and Zannone, N. "Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank," Technical Report DIT-06-002, Department of Information and Computing Technology, Trento University, Trento, Italy, 2006.

Mintzberg, H. *Structuring of Organizations*, Prentice Hall, 1978.

Mintzberg, H. *Structure in Fives: Designing Effective Organisations*, Prentice Hall, 1992.

Moffett, J.D. "Control Principles and Role Hierarchies," 3rd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, November, 1998, pp. 63-69.

Moffett, J.D., Haley, C.B., and Nuseibeh, B. "Core Security Requirements Artefacts," Technical Report 2004/23, Department of Computing, The Open University, Milton Keynes, UK, 2004.

Moffett, J.D., and Lupu, E.C. "The Uses of Role Hierarchies in Access Control," 4th ACM workshop on Role-Based Access Control, Fairfax, Virginia, USA, 28-29th October, 1999, pp. 153-160.

Moffett, J.D., and Sloman, M.S. "The Source of Authority for Commercial Access Control," *IEEE Computer* (21:2), February 1988, pp. 56-69.

Mylopoulos, J., Chung, L., and Nixon, B. "Representing and Using Non-Functional Requirements: a Process-Oriented Approach," *IEEE Transactions on Software Engineering* (18:6), June, 1992, pp. 483-497.

Nuseibeh, B., and Easterbrook, S. "Requirements Engineering: A Roadmap," The Future of Software Engineering, Limerick, Ireland, 4-11th June, 2000, pp. 35-46.

Neumann, G., and Strembeck, M. "Scenario Driven Role Engineering Process for Functional RBAC Models," 7th ACM symposium on Access Control Models and Technologies (SACMAT'02), Monterey, California, USA, 3-4th June, 2002, pp. 33-42.

Park, J.S., Costello, K.P., Neven, T.M., and Disomito, J.A. "A Composite RBAC Approach for Large Organisations," 9th ACM Symposium on Access Control Models and Technologies (SACMAT'04), Yorktown Heights, New York, USA, 2-4th June, 2004, pp. 163-172.

Potts, C. "Using Schematic Scenarios to Understand User Needs," Designing Interactive Systems: Processes, Practices, Methods, & Techniques (DIS'95), Ann Arbor, Michigan, USA, 23-25th August, 1995, pp. 247-256.

Rollinson, D. *Organizational Behaviour & Analysis: An Integrated Approach*, Pearson Education Ltd, 2005.

Ryser, J., and Glinz, M. "Dependency Charts as a Means to Model Inter-Scenario Dependencies," GI-Workshop, Bad Lippspringe, Germany, 28-30th March, 2001, pp. 71-80.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L., and Youman, C.E. "Role Based Access Control Models," *IEEE Computer* (29:2), February 1996, pp. 38-47.

Sandhu, R., Ferraiolo, D., and Kuhn, R. "The NIST model for Role-Based Access Control: Towards a Unified Standard," 5th ACM Workshop on Role-Based Access Control, Berlin, Germany, 26-28th July, 2000, pp. 47-63.

Schaad, A. "A Framework for Organisational Control Principles," PhD Thesis, *Department of Computer Science*, The University of York, York, UK, 2003.

Schaad, A., Moffett, J., and Jacob, J. "The Role-Based Access Control System of a European Bank: A Case Study and Discussion," 6th ACM Symposium on Access Control Models and Technologies (SACMAT'01), Chantilly, Virginia, USA, 3-4th May, 2001, pp. 3-9.

Sim, S.E., Easterbrook, S., and Holt, R.C. "Using Benchmarking to Advance Research: A Challenge to Software Engineering," 25th International Conference on Software Engineering, Portland, Oregon, USA, 3-10th May, 2003, pp. 74-83.

Sindre G., and Opdahl, A.L. "Eliciting Security Requirements by Misuse Cases," TOOLS Pacific 2000, Sydney, Australia, 20-23rd November, 2000, pp. 120-131.

Sindre, G., and Opdahl, A.L. "Templates for Misuse Case Description," 7th Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'01), Interlaken, Switzerland, 4-5th June, 2001.

Spivey, J. *Z-Notation - A Reference Manual*, (2nd Edition), Prentice Hall, 1992.

Strens, M.R., and Dobson, J.E. "Responsibility Modelling as a Technique for Requirements Definition," *IEE Intelligent Systems Engineering* (3:1), 1994, pp. 20-26.

Sutcliffe, A.G., and Maiden, N.A.M. "The Domain Theory for Requirements Engineering," *IEEE Transactions on Software Engineering* (24:3), 1998, pp. 174-196.

Sutcliffe, A.G. "Domain Analysis for Software Reuse," *Journal of Systems and Software* (50:3), 2000, pp. 175-199.

Sutcliffe, A.G., Paparmargaritis, G., and Zhao, L. "Comparing Requirement Analysis Methods for Developing Reusable Component Libraries," *Journal of Systems and Software* (79:2), 2006, pp. 273-289.

Thomas, R.K. "Team-Based Access Control A Primitive for Applying Role Based Access Controls in Collaborative Environments," 2nd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 6-7th November, 1997, pp. 13-19.

Thomas, R.K., and Sandhu, R.S. "Conceptual Foundations for a Model of Task-Based Authorizations," Computer Security Foundations Workshop VII (CSFW 7), Franconia, New Hampshire, USA, 14-16th June, 1994, pp. 66-79.

Uchitel, S., Kramer, J., and Magee, J. "Detecting Implied Scenarios in Message Sequence Chart Specifications," *ACM SIGSOFT Software Engineering Notes* (26:5), 2001, pp. 74-82.

van Lamsweerde, A. "Formal Specification: A Roadmap," The Future of Software Engineering, Limerick, Ireland, 4-11th June, 2000a, pp. 147-159.

van Lamsweerde, A. "Requirements Engineering in the Year 00: A Research Perspective," International Conference on Software Engineering (ICSE'2000), Limerick, Ireland, 4-11th June, 2000b, pp. 5-19.

van Lamsweerde, A. "Elaborating Security Requirements by Construction of Intentional Anti-Models," 26th International Conference on Software Engineering (ICSE'04), Edinburgh, UK, 23-28th May, 2004, pp. 148-157.

van Lamsweerde, A., Brohez, S., Landtsheer, R., and Janssens, D. "From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering," 2nd International Workshop on Requirements for High Assurance Systems (RHAS 2003), Monterey, California, USA., 9th September, 2003, pp. 49-56.

van Lamsweerde, A., Darimont, R., and Massonet, P. "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt.," 2nd International Conference on Requirements Engineering (RE'95), York, UK, 27-29th March, 1995, pp. 194-203.

van Lamsweerde, A., and Willemet, L. "Inferring Declarative Requirements Specifications from Operational Scenarios," *IEEE Transactions on Software Engineering* (24:12), December 1998, pp. 1089-1114.

Weidenhaupt, K., Pohl, K., Jarke, M., and Haumer, P. "Scenario Usage in System Development: A Report on Current Practice," *IEEE Software* (15:2), 1998, pp. 34-45.

Woodcock, J.,Davies J. *Using Z: Specification Refinement and Proof*, Prentice Hall, 1996.

Yao, W., Moody, K., and Bacon, J. "Model of OASIS Role-Based Access Control and its Support for Active Security," 6th Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, 3-4th May, 2001, pp. 171-181.

Yu, E. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," 3rd IEEE International Symposium on Requirements Engineering (RE'97), Annapolis, Maryland, USA, 6-10th January, 1997, pp. 226-235.

Yu, E., and Liu, L. "Modelling Trust in the i* Strategic Actors Framework," 3rd Workshop on Deception, Fraud and Trust in Agent Societies, Barcelona, Spain, 4th June, 2000.

Yu, E., and Mylopoulus, J. "Understanding 'Why' in Software Process Modelling, Analysis and Design," 16th International Conference on Software Engineering (ICSE'94), Sorrento, Italy, 16-21st May, 1994, pp. 159-168.

Zave, P. "Classification of Research Efforts in Requirements Engineering," *ACM Computing Surveys* (29:4), 1997, pp. 315-321.