On Modelling Access Policies:
Relating Roles to their Organisational Context

Robert Crook Darrel Ince Bashar Nuseibeh

Department of Computing, The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
Email: {R.P.Crook, D.C.Ince, B.Nuseibeh} @ open.ac.uk

Abstract

The restriction of access is a mechanism by which organisations protect their information assets. Requirements models use actor definitions to describe users and to specify their access policies. Actors normally represent roles that users adopt, while roles can represent different things, such as a position in an organisation or the assignment of a task. Current requirements modelling approaches do not provide a systematic way of defining roles for incorporation into access policies. We address this issue by proposing a framework that facilitates the derivation of role definitions from their wider organisational context. We illustrate how our framework can be used to extend a formal version of i – to define and verify access policies definitions – and demonstrate its applicability via a case study.*

1 Introduction

Security incidents can be costly; Nick Leeson's trading resulted in losses of over £800 million so causing the bankruptcy of Barings Bank [7], and John Rusnak defrauded the Allied Irish Bank of a similar amount in 2002. They both exploited weaknesses in the computer systems used to the control their trading activities. Thus, while there is a need to keep outsiders from breaking in, there is also a need to prevent users with legitimate access rights not to abuse their privileges. Many organisations have procedural controls, defined as policies, to prevent such abuse. The procedures are often enforced by computer systems, which restrict access. We believe that early understanding and specification of access policies are key to effective access control.

Access policies are rules that specify which users can carry out which actions to enforce principles of management control [35]. In this paper we focus on access policies that enforce one of these principles: that of minimum privileges [2]. This states that users can only access the functions and resources that they require to carry out their duties.

Many requirements models represent users as actors or agents that are assigned to actions. These assignments can be used to represent access policies [9][10][31]. An actor definition usually represents a role rather than a specific person. However, the use of the notion of a role can vary: from the assignment of a task, as proposed by Yu [47], to a position within an organisational hierarchy [37]. Existing approaches to modelling requirements are inadequate for representing complex relationships between actors in large organisations, such as the lines of authority, the organisational structure, and the basis by which work is delegated [39][28]. This can lead to misunderstandings about the precise meaning of actors, their roles, and, consequently, their access rights and privileges. For example, if a doctor is defined as an actor who can read medical records, then the constraint that a doctor only has that role with respect to his own patients cannot generally be represented in existing models.

This missing link between actor definitions and the wider organisational context is the focus of this paper. A key contribution is to demonstrate, through a case study of a large organisation, how access policies satisfying the minimum privileges principle can be specified precisely, using an extended version of Formal Tropos [18]. The paper also shows how such policies can be verified, to ensure that, when enforced, they only allow users to carry out the expected activities that fulfil their duties.

The paper is structured as follows. In section 2 we review the literature on security requirements. In section 3 we examine the security literature concerned with management control and access policies. In section 4 we review the i* framework and a variant known as formal Tropos. In section 5 we introduce our framework for defining access policies and show how the framework can be used to extend formal Tropos with role definitions and specifying access policies. In section 6 we demonstrate the validity of our framework by applying it to a case study. The paper concludes with a discussion and a summary.

2 Modelling of Security Requirements

Security requirements are those requirements concerned with the protection of valuable assets. They arise from the top level security objectives such as maintaining confidentiality, integrity, and availability [8].

To determine security requirements, an important step is the analysis of threats. Researchers have explored threats analysis and the evaluation of countermeasures by extending use cases [40][41][1], by deriving obstacles to goals [27][24], by bounding the scope of security problems [29], and by analysing the social context [46][31]. All these approaches offer systematic approaches for the identification of possible attacks and the definition of countermeasures. None however addresses access policies, which constrain how authority can be exercised and the freedom by which individuals can act.

Fontaine [16] proposes how the assignment of actions to agents maps onto access policies, and presents a formal systematic approach to defining access policies based on KAOS [15]. However, his definition of an agent is ambiguous: it can be a level of authority, a qualification, or a physical individual.

He and Anton [20] propose a systematic approach to deriving roles from scenarios for defining restrictions to satisfy the privacy and security requirements. Their approach identifies tasks that require access to resources that are to be protected, and consequent roles. However, these roles are only derived from tasks, and not the roles that represent other aspects of the organisational context, such as position or level of authority.

Liu *et al.* [31] demonstrate, using the i* framework [47], how the actor boundary in a Strategic Rationale model provides a basis for deriving an access control restriction. An actor boundary encompasses the tasks that an actor carries out and the resources to which that actor has access. i* differentiates between agent definitions (representing physical people) and roles, allowing the modelling of a user adopting multiple roles. As with the above approaches however, what a role denotes is not explicit, so misunderstandings about what a role exactly means are possible.

3 The Principles of Management Control and Access Policies

Access policies provide a means to enforce management controls, controls that may have been in existence long before computer systems. Control principles include clearance levels [5], the separation of duties [12], Chinese walls in the financial services industry [6], supervision and review [34], accounting principles [2], and minimum privileges [2]. Access policies are important components of an overall security policy defined by organisations to enforce these principles., and which ultimately translate into access control mechanisms.

Access policies arise from organisational requirements [45]. The organisational context has been the basis of policy specification languages such as OASIS [4], ASL [26] and Ponder [13], all of which have the notion of roles. The use of roles for defining policies became a focus of research in the 1990´s on Role-Based access control (RBAC). Here, a role is a set of permissions, and can be used to define policies that represent the assignment of tasks in an organisation [37]. Sandhu *et al.* [37] propose that a role inheritance hierarchy can be used to map onto an organisational hierarchy, whereby senior roles inherit the permissions of junior roles within the hierarchy. However, as Moffett observes [34], a manager does not necessarily inherit the roles of his juniors. Moffett and Lupu [33] address this problem by proposing separate hierarchies to capture different aspects of the organisation, such as functional specialisation using an inheritance hierarchy, division of tasks using an aggregation hierarchy, and a supervisory hierarchy to model the seniority of roles with respect to each other.

Contextual factors associated with roles have also been examined. Bertino *et al.* [3] describe how temporal constraints can be defined for roles, while Georgiadis *et al*. [19] combine contextual information with team-based access control. Team-based roles [44] are useful for collaborative working environments, where users are assigned to teams and get access to the team's resources.

Researchers have also explored techniques for deriving roles. Role Engineering, as outlined by Coyne [11], is a systematic process of identifying the activities of a single user and defining this as a role. Fernandez and Hawkins [17] propose deriving roles from use case actor definitions and Neuman and Strembeck [36] propose deriving roles from scenarios of the work-process. This is a bottom up approach to deriving roles from tasks and largely ignores the wider organisational context.

Research into organisational structures also gives insights into the way in which groups are formed. The organisational structure, which is fundamental to management control, includes the allocation of formal responsibilities to interrelated groups and roles. The two key dimensions are *lines of authority*, referred to as vertical differentiation, and *division of work*, referred to as horizontal differentiation [21]. Mintzberg [32] describes two fundamental characteristics by which horizontal differentiation is achieved: functional and market characteristics. Functional characteristics include the division of work on the basis of function, qualification, and work process. Market characteristics include organisational division based on customers, service, product, location, or time; functions are replicated but the market for which the organisational division or unit is responsible differs. In large organisations, several of these characteristics are often used. The National Health Service in the UK, for example, is divided into regional health authorities that, in turn, are composed of hospitals to serve the different population centres, so that the authority and

the hospitals are organised on a geographical basis. A hospital, however, is organised on a functional basis. Similarly, retail banks have autonomous branches dispersed to serve local markets, with an identical functional structure in each branch.

The mid 1990's saw requirements engineering (RE) research relating goals to organisational context. For example, ORDIT [14] focused on the delegation of responsibilities to agencies, rather than the structural organisational relationships. In contrast, the teleological approach of Loucopoulos and Kavakli [28] focused on deriving goals from the organisational activities, but did not clarify the lines of authority and delegation. However, with the exception of i* described in the next section, other RE approaches have largely ignored the organisational context.

4 The i* Framework and Formal Tropos

We focus on extending the i* framework because of its relevant focus on the intentions of actors in a social context. The organisational context is closely related to the social context, in that an organisation is a group of individuals that relate to one another. In modelling the social context, i* goes further than other approaches in modelling relationships between actors.

Liu *et al*. [31] propose the use of i* Strategic Rationale (SR) models for defining policies, and suggest using the actor boundary on which to base a policy. From this, a constraint can be defined in an RBAC access control system, such that the role is derived from the actor definition and the permissions are derived from the tasks within the boundary. An example SR model that illustrates this is shown in Figure1. The actor in the diagram is a Family Doctor. The goal of this actor is to provide a regular clinical service. In order to achieve this goal, the actor needs to open new medical records. Open new medical record is therefore defined as a task dependency of the goal, and a means-end link relates this task to the resource medical record. Family Doctor is defined as an agent, which is a type of actor. Liu *et al.* also give an example of an instantiation of this SR diagram, where an instantiation of an agent Dr. Jones can access the medical record of Mr. Smith. Resources within an actor boundary that represents an instantiation, are also themselves instantiations. The medical record of Mr. Smith is an instantiation of medical record. This provides a useful means of verifying a policy. In effect, this instantiation is a scenario, and since stakeholders can relate more easily to scenarios then abstract definitions, scenarios can be used to elicit and validate requirements [22].

An actor in i* can be an agent, role, or position. An agent is a physical entity such as a human, a role is 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.

Formal Tropos [18] formalises i* in order to allow model checking of its descriptions. Formal Tropos describes the relevant objects of the modelled domain and has two layers. The outer layer models the classes, which can be an actor, a dependency, or an entity. Entities do not exist in i* and are used to represent elements that do not appear in the model as they are not directly related to actors' strategic goals. Attributes in the class definitions represent relationships between classes. The inner layer of the formal Tropos language is a first order predicate language with temporal constraints. In formal Tropos, an example of an actor assigned to a goal is as follows, taken from the SR diagram in figure 1:

Actor Family Doctor
Goal Provide Regular Clinical Service
Mode Achieve

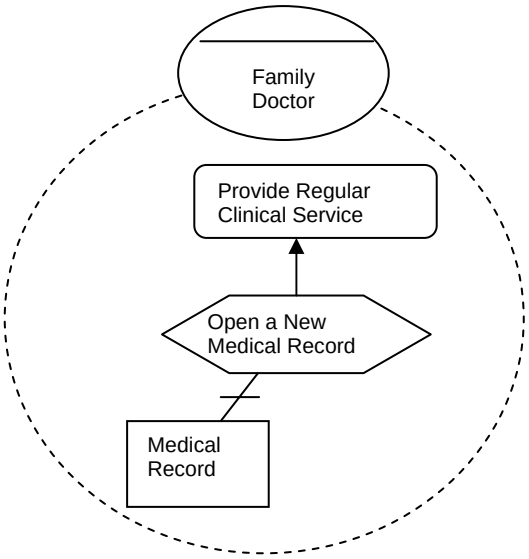

Figure 1: Strategic Rationale Diagram

Any tasks or resources related to the goal can be represented within a fulfilment condition, using the first order predicate language. Instantiation and verification is achieved by translating the model into an intermediate language, which is then interpreted by the model checking tool NuSMV to automatically instantiate objects and ensure model constraints are enforced. We will be deviating from this philosophy somewhat, but adhering to the approach proposed by Liu *et al.* using i*, where tasks and resource are explicitly defined and related to the goal within an actor boundary. We will also be adopting Liu *et al.'s* approach to instantiation, where the modeller explicitly defines a scenario to verify the more abstract definitions. We will therefore be using the same declarative notation but with some modifications.

5 A Framework for Modelling Policies

We now present our framework for modelling access policies. It consists of a meta-model that describes domain independent abstractions – which we refer to as meta-concepts – and how they relate to one another. The framework enables policies to be 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 Hospital Doctor, 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, and Dr Smith is an instantiation of an 'agent'.

We present our framework formally using the state-based Z notation [42]. This reduces ambiguity and has allowed us to check policies developed using our framework. A state-based notation is appropriate for our purposes because we are concerned with maintaining a secure state of the system for any arbitrary point in time. State-based formal methods are usually associated with proof tools. However, performing proofs – even with the support of a tool – can be an arduous process [43]. So, we have adopted a lightweight formal modelling language and tool, Alloy [25], which is a subset of Z, to compile and verify our models. Fuxman *et al*. [18] highlight the benefits of using tools to verify requirements models; in particular they assist in checking for inconsistencies and animating specifications.

We first introduce our framework's meta-model elements: the domain independent abstractions in access policy specifications. These include roles and organisational structural elements of organisational domains, organisational functions, and levels of authority. We then relate roles to the organisational contextual elements at the meta and domain levels. We elaborate on how aggregation and inheritance between these organisational contextual elements can be used to represent characteristics of the organisational structure. We then explain how tasks and assets relate to one another. We then suggest how access policies can be defined and verified, and show how, through the formal Tropos notation, the i* framework can be extended based on the concepts of our policy framework. Finally, we show how a model in formal Tropos can be translated into our framework and verified.

5.1 Framework 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 an *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 wider organisational context, we also need some additional meta-concepts:

[*org_function*] represents a functional grouping within an organisation. Members of a functional grouping will be expected to carry out tasks that will be delegated this group.

[*org_domain*] represents a "market based" grouping i.e. a grouping that is delegated a market to serve, such as a set of clients in a specific geographic location. An example of this would be a branch in a bank, which serves customers in their locality.

[*authority*] represents the seniority of a role.

5.2 Relating Roles to the Organisational Context

The main objective of our framework is to enable the definition of roles that are linked to their wider organisational context with respect to the two key dimensions by which work is differentiated within organisations. These are the lines of authority (vertical differentiation) and the division of work (horizontal differentiation). We can capture this through the three framework elements: [*authority*] representing a level of authority, [*org_function*] representing the differentiation of functions, and [*org_domain*] representing differentiation according to market based characteristics. An element role is assigned a product of these three types:

$role = authority \times org\_function \times org\_domain$

A role thus represents a position in an organisation, for example a role representing a nurse. A nurse has a level of seniority, independent of the medical speciality in which he works. The nurse is assigned to a medical speciality, such as orthopaedics, which in our role definition is represented as an organisational function. Finally the nurse executes his function in a ward or operating theatre, which is represented as an organisational domain.

5.3 Inheritance and Aggregation Hierarchies

Our framework includes inheritance hierarchies in order to represent appropriate levels of abstraction. For example, we can define medical practitioner as a role and surgeon or physician as specialisations of this. We model this as:

$inhf : org\_function \rightarrow org\_function$

Moffett and Lupu [33] explain the usefulness of an inheritance hierarchy for organisational functions. In a hospital, for example, there are many types of medical specialists. It is more efficient to define a single policy restricting the access of medical records to medical specialists rather than specific ones for the different types of physicians and surgeons. The inheritance of roles is represented by the following function:

$inhr : role \rightarrow\!\!\!\!\!\!\bullet\ role$

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

$role2 \in \mathbf{ran}\ inhr^* \lhd role1$

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.

For organisational domains, an aggregation hierarchy can also be useful, in order to capture the subdivision of markets [32], such as subdividing bank regions into local branches. This is expressed as:

$aggd : org\_domain \rightarrow\!\!\!\!\!\!\bullet\ org\_domain$

5.4 Levels of Authority

The meta-concept 'level of authority', as part of a role, represents the seniority of that role. If we want to represent the organisational hierarchy as proposed by Moffet and Lupu, 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:

$senior : authority \rightarrow\!\!\!\!\!\!\bullet\ authority$

Seniority is a function that 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 [21] and hence report to more than one superior. This can be represented in this framework by assigning an agent multiple roles in different organisational domains.

5.5 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, where aggregation can be modelled as:

$aggt : task \rightarrow\!\!\!\!\!\!\bullet\ \mathbb{P}\ task$

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:

$task\_asset\_dependency : task \rightarrow \mathbb{P}\ asset$

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

$asset\_domain : asset \rightarrow org\_domain$

5.6 Policy Definitions

We define policies using an *authorisation_policy* function:

$authorisation\_policy \cong role \times task$

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*.

5.7 Policy Verification through instantiation

In the previous section we described the example proposed by Liu *et al.* [31], where an instantiation of an agent was used to verify a requirement. We now explain how our own framework 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 an action. 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 a function for a medical specialist, then this function will not change for the instantiation, nor for the level of authority. Instantiations are required of organisational domains, roles, assets, and agents.

An organisational domain instantiation will represent a specific organisational unit. For example, if a bank has branches, a branch is a domain description of an organisation unit, but the Sheffield branch 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 customer advisor of a bank as an abstract role, then an instantiation of this would be a customer advisor in the Sheffield branch.

In order to represent instantiation, we define a number of functions. For simplicity, we do not use separate types to represent instantiations; an earlier version of our framework [9] did include them and resulted in invariants that were difficult to read and manipulate. For organisational domains the instantiation is:

$insd : org\_domain \rightarrow\!\!\!\!\!\!\bullet\ org\_domain,$

where the domain *org_domain* is instantiated from the range, and likewise the instantiation for roles is:

$insr : role \rightarrow\!\!\!\!\!\!\rightarrow role$

Instantiated roles are assigned to agents, which represent humans:

$assigned\_role : agent \rightarrow\!\!\!\!\!\!\rightarrow \mathbb{P}\, role$

We also need to define a task execution that represents the carrying out of a task on a specific instance of an instance. This we represent via a function *performs*, which defines an agent performing a task on an asset:

$performs: agent \rightarrow task \times \mathbb{P}\, asset$

For convenience we can define the act of performing a task on a set of assets as a *task_execution*:

$task\_execution : task \times \mathbb{P}\, asset$

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

$\forall user : agent, \forall\ user\_task : task\_execution \bullet user\_task \in perfoms\ (user) \Rightarrow \exists role : assigned\_role\ (user) \bullet \exists policy :$
$authorisation\_policy \bullet policy.role \in \mathbf{ran}\ inhr^* \lhd insr(role)$
$policy.task = user\_task.task$
$\forall asset : user\_task, asset \bullet asset\_domain(asset) = role.org\_domain$

This invariant is defined in the form: $P \Rightarrow Q$. P is the assertion that an agent has executed a task (though P can be a set of mappings between agents and task executions), 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 defined in the policy must be equivalent to the task in the *task_execution.* Third, the assets being accessed through the task execution must be in the same organisational domain as the user.

The invariant is therefore a check on the *performs* function, which contains all mappings between agents in the system and task executions, i.e. tasks they have executed. If we define a mapping between an agent and a task execution in the *performs* function, the invariant tells us whether it is permissible. If the invariant is true, then the task execution could be performed by that agent. There are similarities between an instantiated specification as we have presented it and an RBAC system. The key difference however is that our instantiated specification contains only organisational phenomena; an RBAC system in contrast is a computerised implementation.

5.8 Framework Invariants

There are a number of assumptions in the framework that should be defined as invariants. For example, a role can not inherit itself and only instantiated roles can be assigned to agents. Due to limitations of space we have not included them in this paper.

5.9 Extensions to Formal Tropos

Having outlined the conceptual elements of our framework, we now consider how to incorporate them into Formal Tropos. Associated tasks and resources are defined below, using an indentation to represent the means-end to a goal, and the resource dependency relationship. We have added a type attribute to actor to enable us to differentiate between agents, positions, and roles. Inheritance, aggregation, and instantiation between domain elements, are represented by using the keywords ISA, Part, and INS respectively, as used in i*.

Actor Customer Advisor
  Type Agent
    Goal Provide Credit
    Mode Achieve
      Task Evaluate Credit
        Resource Credit Conditions
        Resource Credit Application
        Resource Credit History

Next we consider the modelling of roles and associated organisational characteristics. Referring back to the example of Liu *et al.*, Family Doctor was defined as an agent and Dr. Antony as an instantiation of that agent. In this particular example, the abstract agent and instantiation of an agent in i* correspond to a role and an agent, respectively, in our framework. As discussed earlier, an actor can alternatively be a position or a role.

A role in the i* framework has a very specific meaning in that it represents an assigned task. Hence it is not appropriate to use here. One could possibly define it as a position in i*, but we would still need to assign this to some agent, adding to the complexity of the model. We have therefore decided to map role definitions of our framework onto abstract agent definitions.

We need to link agent definitions to the organisational contextual elements, level of authority, organisational function, and organisational domain, which we can define as classes in Formal Tropos. So, for example, we can define the agent of Family Doctor as:

Actor Family Doctor
  Type Agent
  Organisational Function General Practice
  Organisational Domain Practice
  Authority Consultant
    Task Administrate Medical Record
      Resource Medical Record

This represents an extension to the authorisation policy that we presented in section 4. We have added a level of authority to the definition, reflecting a seniority grade that exists within the National Health Service in the UK, and the organisational domain Practice, the domain in which the Family Doctor executes his function. In order to verify the policy, we must not only instantiate the agent but also

the organisational domain and resource. An instantiation of the domain is defined as:

Organisational Domain Dr Antony´s Practice **INS** Practice

Since the policy involves the access to a medical record, we also need an instantiation for that:

Resource Medical Record of Jim Smith **INS** Medical Record

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

Actor Dr Anthony **INS** Family Doctor
Type Agent
Organisational Function General Practice
Domain Dr Antony´s Practice **INS** Practice
Authority Consultant
Task Administrate Medical Record
Resource Medical Record of John Smith **INS** Medical Record

6 Case Study

To validate our framework, we used a case study from the literature [38] that explores several principles of management control, including the minimum privileges, 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.

The case study is based on an access control system of a European Bank. The bank has 50,000 employees, over thousand branches, and provides banking services for local communities. Schaad [38] 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 within the constraints of this paper, 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 two of these services: the provision of a fixed sum of money and share trading. These two are mutually exclusive subject to the principle of separation of duties. Although we will not formally define a policy, we discuss how role definitions as we have made them can help in formulating such a policy. The flow diagram in figure 2 shows some of the steps involved.

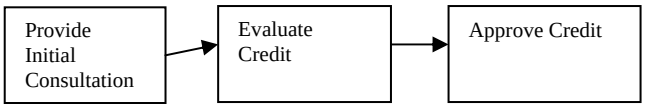

Figure 2: Basic credit application process

This 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 advisor clerks. The approval of credit is done by the advisor's manager. The functions customary advisory services are 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 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 customary advisory services and share trading within a branch.

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 3. 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 domains and organisational functions that are as follows:

Organisational Function Customary Advisory Services

Organisational Function Share trading

The domains are the regional domain and for each branch:

Organisational Domain Region
Organisational Domain Branch
Part Region

Within each grouping there is a hierarchical structure. Focusing on the branch customary advisory services, 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 customary advisory services:

Actor Customer Advisory Services Manager
Type Agent
Organisational Function Customary Advisory Services
Organisational Domain Branch
Authority Manager

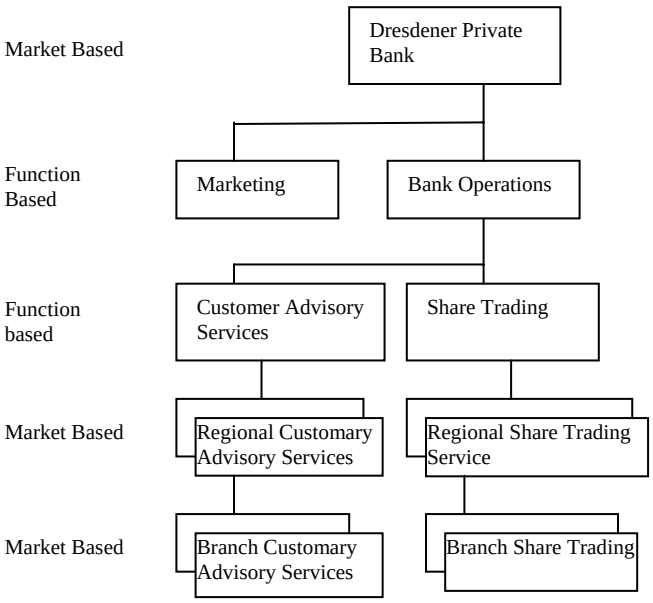

Figure 3: Organisational Structure of the Dresdener Bank

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

Task Initial Consultation
  ResourceCredit Contract

Task Evaluate Credit
  Resource Credit Contract, Credit History

Task Approve Credit Contract
  Resource Credit Contract

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

Actor Customer Advisory Services Manager
 Type Agent
 Organisational Function Customary Advisory Services
 Organisational Domain Branch
 Authority Manager
  Task Approve Credit

Actor Customer Advisory Services Clerk
 Type Agent
 Organisational Function Customary Advisory Services
 Organisational Domain Branch
 Authority Clerk
  Task Initial Consultation, 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 business advisory services, 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 our framework. The definition of the authority of Clerk is:

Clerk : authority
senior : {*Clerk* ↦ *Manager*}

Similarly, definitions exist for the other authority levels, organisational functions, organisational domains, tasks, and resources, which we will not repeat here. An example of one of the role definitions is:

CustomerAdvisoryServicesClerk.org_function = CustomerAdvisoryServices
CustomerAdvisoryServicesClerk.org_domain = Branch
CustomerAdvisorServicesClerk.authority = Clerk

The above role definition is a mapping from the Formal Tropos actor definition of a Customer Advisory Services Clerk. One of the policies related to this definition, in order to restrict the use of Initial Consultation, is defined as:

InitialConsultationPolicy : *authorisation_policy*
InitialConsultationPolicy.task = InitialConsultation
InitialConsultationPolicy.role=CustomerAdvisoryServicesClerk

Similarly, for the other tasks such as *Evaluate Credit* and *Approve Credit Conditions*, 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 Clerk can evaluate credit and alter the credit conditions of a customer of the branch to which he is assigned. First, we define a domain instantiation for the Frankfurt branch:

Organisational Domain Frankfurt Branch **INS** Branch

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

Actor Jim Smith **INS** Customer Advisory Services Clerk
 Type Agent
  Organisational Domain Frankfurt Branch
  Organisational Function Customer Advisory Services
  Level of Authority Clerk

These two Tropos definitions map onto the policy:

FrankfurtBranch : *org_domain*
CustomerAdvisoryServicesClerkFrankfurt : role

The following definition shows that Frankfurt Branch instantiates the Branch, i.e. it is a branch:

insd : { *Frankfurt Branch* ↦ *Branch*}

The following definitions represent the instantiated role for a Customer Advisory Service Clerk in the Frankfurt Branch:

CustomerAdvisoryServicesClerkFrankfurt.authority = Clerk
CustomerAdvisoryServicesClerkFrankfurt.org_function =CustomeryAdvisoryServices

CustomerAdvisoryServicesClerkFrankfurt.org_domain = Frankfurt Branch

insr : {*CustomerAdvisoryServicesClerkFrankfurt* ↦ *CustomerAdvisoryServicesClerk*}

We also need to define the instantiation of assets. So we define the assets credit application and credit history of the customer Philip Stokes. We assign these assets to the Frankfurt branch:

Resource CreditApplication of Philip Stokes **INS** CreditApplication
Organisational Domain Frankfurt Branch

Resource CreditHistory of Philip Stokes **INS** CreditHistory
Organisational Domain Frankfurt Branch

These definitions then translate into our policy framework. First, we create the asset affected:

CreditApplication of Philip Stokes, CreditHistory of Philip Stokes : *asset*

We then define them as instantiations:

insa : { *CreditApplicationOfPhilipStokes* ↦ *CreditApplication, CreditHistoryOfPhilipStokes* ↦ *CreditHistory* }

EvaluateCredit, *AlterCreditConditions* and *ApproveCreditConditions*

We define *performs* as follows:

performs = { *JohnSmith* ↦ *InitialConsultation* x *CreditApplication of Philip Stokes*,
JohnSmith ↦ *EvaluateCredit* x {*CreditApplication of Philip Stokes, CreditConditions of Philip Stokes }*}

Since this *performs* definition satisfies the invariant definition described in section 5.7, the instantiation is consistent with the policy. What we could also try is an instantiation of an agent who should not be able to perform a task on a certain asset, such as Customer Advisory Service Clerk in another branch accessing a credit application of a customer not in his branch. The invariant definition would then be false.

The way in which we have defined roles separating the level of authority, organisational function, and organisational domain is very similar to the way in which the Dresdener Bank has actually designed its access control system [38]. Roles in the Dresdener Bank RBAC system are composed of a function, a position, and an organisational unit. A function reflects an organisational function in our framework, a position reflects a level of authority, and the organisational unit reflects an organisational domain. Thus, definitions derived in the way would map easily onto Dresdener's RBAC system. Although there are similarities, these are coincidental. An earlier version of the framework was developed around an example in the medical domain [9].

7 Summary and Conclusions

This paper has 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 a macro-organisational analysis before specifying actor or role definitions in the context of modelling of access policies. The lack of this in current modelling approaches makes it difficult to unambiguously express access policies and to refine them into operational constraints. We proposed a novel way of deriving roles from the macro-organisational context. We demonstrated how to identify the groupings, the levels of authority, and the management domains from which role can be defined. It is relating roles to identifiable phenomena of the organisational context, i.e. the levels of authority and groupings, that gives us a more precise definition.

A key contribution of our work is that we demonstrated how access policies satisfying the minimum privileges management control principle can be specified unambiguously and verified using an extended version of Formal Tropos. However there are other principles that remain to be investigated. Accounting principles, for example, can lead to complex procedures, whereby workflows need to be modelled and financial constraints such as credit ratings need to be included in the policies.

In addition to extending Formal Tropos, there are other avenues worth investigating. In particular we need to combine our approach with the other approaches identified in section 2 with regard to threats and countermeasures analysis.

Acknowledgements. Thanks Jonathan Moffett and Qingfeng He for their helpful comments on an earlier draft of the paper. Nuseibeh was supported by grants from the Leverhulme Trust and the Royal Academy of Engineering.

Reference

[1] I. Alexander, "Modelling the Interplay of Conflicting Goals with Use and Misuse Cases", *Proc. of 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ-02)*, Essen, Germany, 9-10 Sept. 2002.

[2] R. Anderson, *Security Engineering - A Guide to building dependable distributed System*s, Wiley, 2001.

[3] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: Temporal Role-based Access Control Model", *Proc. of 5th ACM Work. on Role-based access control*, 21-30, Jul 2000.

[4] J. Bacon, M. Lloyd, and K. Moody, "Translating Role-based Access Control within Context", *Proc. of Int. Workshop Policies for Distributed Systems and Networks (Policy 2001)*, Bristol, UK, Jan. 2001, Springer, 107-119.

[5] D. Bell, and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model*, MITRE Tech Rep. 2547, Vol 2, 1973.

[6] D.F.C Brewer, and M.J. Nash, "The Chinese wall security policy", *Proc. of the IEEE Symposium on Security and Privacy*, 206-214, 1989.

[7] S. J. Brown and O. W. Steenbeek, "Doubling: Nick Leeson's Strategy", *Pacific Basin Journal*, 2001.

[8] British Standards Institution, *BS799-1:1999 Information security management - Part 1: Code of Practice for Information Security*, London, 1999.

[9] R. Crook, D. Ince, and B. Nuseibeh, "Towards an Analytical Role Modelling Framework for Security Requirements", *Proc. of the 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ02)*, Essen, Germany, 2002.

[10] R. Crook, D. Ince, and B. Nuseibeh, "Modelling Access Policies Using Roles in requirements Engineering", *Information and Software Technology*, 45(14): 979-991, November 2003, Elsevier.

[11] E.J.Coyne, "Role Engineering", *Proc. of 1st ACM Workshop on Role-based access control*, 1996.

[12] D.D Clark, and D.R Wilson, "A comparison of commercial and military computer security policies", *Proc. of IEEE Symposium on Security and Privacy*, 184-194, 1987.

[13] N. Dulay, E. Lupu, M Sloman, and N. Damianou, "A Policy Deployment Model for the Ponder Language", *Proc. IEEE/IFIP Intl. Symposium on Integrated Network Management (IM'2001)*, Seattle, USA, May 2001.

[14] J.E. Dobson and M.R. Strens, "Organisational Requirements Definition for Information Technology Systems", *Proc. of 1st IEEE Int. Conference on Requirements Engineering (ICRE '94)*, 158-165, Colorado Springs, Colorado, USA,18-22 April 1994.

[15] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, Volume 20, 1993.

[16] P. Fontaine, "Goal oriented elaboration of security requirements", *Project Dissertation*, Universite Catholique de Louvain, Belgium, 2001.

[17] E.B. Fernandez, and J.C. Hawkins, "Determining role rights from Use Cases", *Proc. of 2nd ACM workshop on Role-based access control*, 121-125, 1997.

[18] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso, "Model Checking Early Requirements Specifications in Tropos", *Proc. of 5th IEEE Int. Symposium on Requirements Engineering Conference (RE'01)*, Aug 2001, Canada.

[19] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas, "Flexible team-based access control using contexts", *Proc. of 6th ACM Symposium on Access control models and technologies*, 21-27 , May 2001.

[20] Q. He and A. I. Antón, "A Framework for modeling privacy requirements Role Engineering", *Proc. of 9th Intl. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, 137-146, Austria, 16-17 June 2003.

[21] C. Handy, *Understanding Organisations*, Penguin, 1985.

[22] A. Lamsweerde "Requirements Engineering in the year 00: A Research Perspective", *Proc. of Int. Conf. on Software Engineering (ICSE'2000)*, Ireland, June 2000, ACM Press.

[23] A. Lamsweerde, "Formal Specification, A Roadmap", *ICSE-2000 Future of Software Engineering*, A. Finkelstein (Ed.), ACM Press, June 2000.

[24] A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models", *Proc.26th Int. Conference on Software Engineering (ICSE-04)*, 148-157, Edinburgh, UK, May 2004.

[25] D. Jackson , *Micromodels of Software: Lightweight Modelling and Analysis with Alloy Software*, Design Group. MIT Lab. for Computer Science, 2002.

[26] S. Jajodia, P Samarati, and V.S. Sabrahmanian, "A Logical Language for Expressing Authorisations", *Proc. of IEEE Symposium on Research in Security and Privacy*, 31-42 Oakland, USA, May 1997.

[27] A. Lamsweerde, S. Brohez, R. Landtsheer and D. Janssens, "From System Goals to Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering", *Proc. of 2nd Int. Workshop on Requirements for High Assurance Systems (RHAS 2003)*, Monterrey, USA. September 2003.

[28] P. Loucopoulus and E.Kavakli, "Enterprise Modelling and the Teological Approach to Requirements Engineering", *Int. Journal of Intelligent and Cooperative Information Systems*, 4(1): 45-79, 1995.

[29] L. Lin, B. Nuseibeh, and D Ince, "Using Abuse Frames to Bound the Scope of Security Problems", *Proc. of 3rd Int. Works. on Requirements for High Assurance Systems*, 2004.

[30] L. Liu, E. Yu, and J. Mylopoulos, "Analyzing Security Requirements as Relationships Among Strategic Actors", *2nd Symposium on Requirements Engineering for Information Systems (SREIS'02)*, 2002.

[31] L. Liu, E. Yu, and J. Mylopolous, "Security and Privacy Requirements Analysis within a Social Setting", *Proc. Of 11thIEEE Int. Conference on Requirements Engineering (RE'03)*, Monterrey, USA, 2003.

[32] H. Mintzberg, *Structure in Fives: Designing effective organisations*, Prentice Hall, 1992.

[33] J. D. Moffett, E. C. Lupu, "The uses of role hierarchies in access control", *Proc. of 4th ACM workshop on role-based access control*, 153-160, October 1999.

[34] J. D. Moffett, "Control principles and role hierarchies", *Proc. of 3rd ACM workshop on Role-based access control*, 63-69, October 1998.

[35] J. D. Moffett and Morris S. Sloman, "The Source of Authority for Commercial Access Control", *IEEE Computer 21(2)*: 59-69.1988.

[36] G. Neumann and M. Strembeck, "A scenario driven role engineering process for functional RBAC models", *Proc. of 7th ACM symposium on access control models and technologies*, 33-42, 2002.

[37] R. Sandhu, E. Coyne,H. Feinstaein, C. Youmann, "Role Based Access Control Models", *IEEE Computer*, 29(2): 38-47, Feb. 1996.

[38] A. Schaad, "Framework for Organisational Control Principles", *PhD Thesis*, Dept. of Computer Science, University of York, UK, 2003.

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

[40] G. Sindre and A. L. Opdahl, "Eliciting Security Requirements by Misuse Cases", *Proc. of TOOLS Pacific 2000*, 120-131, 20-23 Nov. 2000.

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

[42] J. Spivy, *The Z-Notation - A Reference Manual*, 2nd Edition, Prentice Hall, 1992.

[43] S. Stepney, "A Tale of two Proofs", *BCS-FACS 3rd Northern Formal Methods Workshop*, Ilkley, UK, Sept.'98.

[44] R.K. Thomas, "Team-Based Access Control: A Primitive for Applying Role Based Access Controls in Collaborative Environments", *Proc. of 2nd ACM workshop on Role-based access control*, Fairfax, USA 1997.

[45] R.K.Thomas and R.S. Sandhu, "Conceptual foundations for a model of task-based authorizations", *IEEE Proc. on Computer Security Foundations Workshop VII*, CSFW 7, 66-79, 1994.

[46] E. Yu and L. Liu Modelling, "Trust in the i* Strategic Actors Framework", *Proc. of 3rd Workshop on Deception, Fraud and Trust in Agent Societies*, 2000.

[47] E. Yu, "A Framework for Organizational Modeling", *PhD Thesis*, Department of Computer Science, University of Toronto, Canada, 1995.