A Framework for Organisational Control Principles

Andreas Schaad
University of York
Department of Computer Science
Y010 5DD, York
United Kingdom
Email: andreas.schaad@cs.york.ac.uk

Jonathan D. Moffett
University of York
Department of Computer Science
Y010 5DD, York
United Kingdom
Email: jonathan.moffett@cs.york.ac.uk

Abstract

Organisational control principles, such as those expressed in the separation of duties, supervision, review and delegation, support the main business goals and activities of an organisation. Some of these principles have previously been described and analysed within the context of role- and policy-based distributed systems, but little has been done with respect to the more general context they are placed in and the analysis of relationships between them.

This paper presents a framework in which organisational control principles can be formally expressed and analysed using the Alloy specification language and its constraint analysis tools.

1 Introduction

Recently a new interest in topics such as delegation and revocation of roles and policies, separation of duties and supervision and review could be observed within the context of role- and policy-based distributed systems management [1], [2]. We believe that these concepts can be viewed as belonging to a more general set of organisational controls, showing certain characteristics and dependencies when analysed on their own and in combination.

This paper presents our current approach, intermediate results and lessons learnt in the definition of a framework suitable to express and analyse a set of selected control principles. In line with the framework it focuses on two main aspects: A review and discussion of organisational control principles, their origin, and relationships; and a formal object model to express static constraints and define dynamic, state-changing operations over sequences of states. Technically, this model is supported by using the Alloy specification language and its analysis facilities [3].

The rest of this paper begins with a definition and discussion of organisational control and control principles (Section 2), followed by an outline of the conceptual model underling the framework (Section 2.1). In combination with a modular specification architecture (Section 2.2), this model is suitable to express and analyse a set of control principles. Before doing so, the most fundamental parts of the Alloy language are introduced (Section 2.3), together with an explanation of how to model object behaviour over sequences of states and maintain a history of state changing operations. Some selected separation controls are then specified (Section 3.1), followed by the definition of delegation and revocation controls for authorisation and obligation policies (Section 3.2). The delegation of obligations will reveal the necessity for review controls (Section 3.3.), which are part of more general supervision controls (Section 3.4). The paper finishes with an informal discussion of some results obtained through the dynamic analysis of composed control principles (Section 4).

2 A Control Principle Framework

One way of viewing organisations and the context of this work is the *formal organisation*, defined by an explicit structure linking positions, roles and principals as well as specific relationships, rules and regulations constraining the behaviour of these entities. This view does not, however, imply that this structure is static. It continuously evolves in order to satisfy its goals and this re-organising process is what results in an organisation.

In [4] the major dimensions of organisational structure are described. From these, four key aspects of an organisation can be derived:

- Division of work by grouping together sections, departments, divisions and larger units.
- Definition of roles and positions through allocation of individual tasks and responsibilities, job specialisation and job definition;.
- Formal relationships between organisational entities, levels of authority and spans of control.
- Mechanisms for the delegation of authority and procedures for monitoring and evaluating the use of discretion.

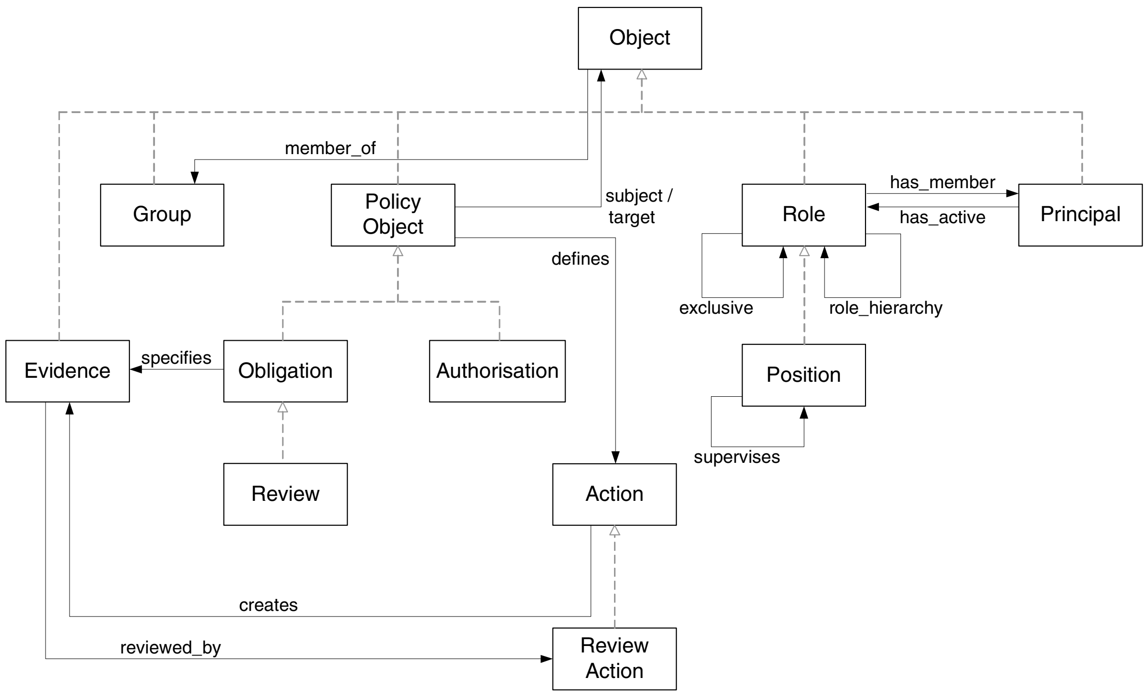

Figure 1: Conceptual Control Principle Model

It is a commonly accepted view to see organisations as goal attainment systems. Top-level goals are refined into a set of business activities. Organisations try to achieve control over these business activities. Control is the means by which activities and resources are coordinated and directed towards the achievement of the goals they have been derived from. This implies a degree of monitoring and feedback, achieved by implementing a system of internal control [5]. Such a system usually follows a control loop of assessing systems and procedures; establishing the appropriate controls; evaluating outputs; and adjusting the controls if necessary. Implementations of control systems will look different from one organisation to another, but there are some underlying control principles that are common to many organisations.

Control principles are general constraints placed upon an organisation and its systems. They are derived from the organisation's top-level goals and are refined together with these goals through the organisational hierarchy. While such principles have been identified long before the integration of computer systems into organisations [6], some of these have received renewed attention within the context of role- and policy-based distributed systems management. These are the principles of:

- Separation of Duties
- Review and Supervision
- Delegation and Revocation

2.1 Underlying conceptual model

It is evident that the conceptual model presented in the following re-uses concepts established by existing role- and policy-based distributed systems management models, more specifically [1] and [2]. In fact, it would be worrying if this were not the case and our model is intended to generalise these with respect to the representation of controls. The structure of the model for control principles that we use as the basis for our specification and later analysis is displayed in figure 1. Grey, dotted lines indicate object extension ("is-a") as supported by the Alloy language, black solid lines declare the relations between the objects. This representation gives only a first overview of the most basic elements and relationships and must not be mistaken for the entire model. Nevertheless, this graphical approach provides a better means of communicating parts of the conceptual model than pure text and maps uniquely to the textual Alloy specification.

Objects can be members of groups. A group is itself an object and thus may also be a member of some other group. A principal is an object representing a human user or automated component in the system. A policy object is an abstraction determining the behaviour of principals in a system and can be either an authorisation or obligation. Policy objects have subjects and targets. While any meaningful object may serve as a target, only roles and principals are valid subjects. For example, an authorisation may have a role and/or a principal as its subject. In other words, a principal may hold a policy object directly or through a role he is a member of. The reasons for this abstraction are implementation specific.

Policy objects define a set of actions. In the case of obligations these are the actions that have to be performed and in case of authorisations the allowed actions. Execution of an action may create evidence which is specified in an obligation such that it can be investigated whether the obligation was satisfactorily met. A review obligation (see section 3.3) is a specific kind of obligation, which has another obligation as its target. A review results from the prior delegation of this target obligation. Review actions are a specific kind of action, which review the evidence generated by the target obligation in order to assess whether it has been successfully discharged.

Roles are a structuring mechanism to relate principals and policy objects. Two role specific relations allow the formation role hierarchies and the definition of mutually exclusive roles. A position is a specific kind of a role with some associated, context-dependent, attributes. Positions can form supervision hierarchies over the supervises relation. Role activation is modelled through a simple activation relation between a principal and a role.

2.2 A specification architecture

The concept of modules in Alloy allows for the separation of the specification into separate, partially dependent packages (Figure 2).

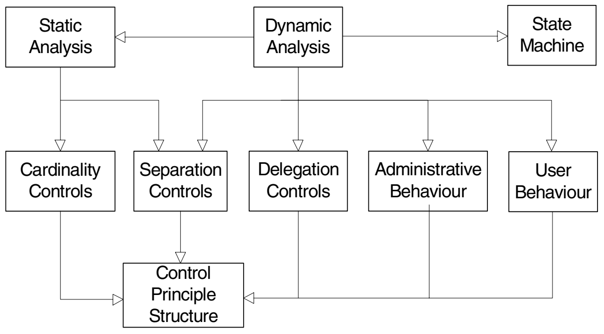

Figure 2: Specification architecture

Each package may contain a set of Alloy signatures, facts and functions. The arrows indicate a dependency relationship between the packages, e.g. the user behaviour package requires the structure package. The purpose of each package can be summarised as follows: The control principle structure package contains the basic objects and relations as described in the previous section. Cardinality constraints such as on the span of control and role activation are part of the respective package. Static and dynamic separation controls such as separation of duties are defined in the separation controls package. Delegation and revocation controls as well as review mechanisms are also defined in a separate package. We further distinguish between user behaviour (e.g. accessing an object) and administrative behaviour (e.g. role hierarchy manipulations). The static analysis package defines a set of assertions about a state while the dynamic analysis package does the same but for sequences of states. The state machine package specifies a mechanism for defining such sequences.

2.3 The Alloy specification language

Alloy is a lightweight first-order logic modelling language. For a full introduction to the language, refer to [3]. It supports a structured specification, which may consist of the following main components:

- *Signatures* - A signature represents a basic type and signature extension is a powerful feature to support hierarchical specification.
- *Fields* - A field relates basic types and is defined within the signature.
- *Facts* - Facts are explicit constraints on the model.
- *Functions* – Functions can be used to relate system states, define specific constraints or return values.
- *Assertions* - Assertions are statements that are supposed to be true. Counterexamples are generated by the analysis tool if an assertion does not hold.

Alloy is supported by a tool for fully automated syntactic and semantic analysis that uses standard satisfiability (SAT) solving algorithms. It has been this tool support which motivated our use of this particular language instead of other specification languages such as Z or the Object Constraint Language OCL. We believe that obtaining immediate feedback from the tool is helpful to catch conceptual and syntactical errors; resolve ambiguities; and generally assist in an explorative approach to model organisational controls. However, we also found that it is very tempting to use Alloy and its tool like a procedural programming language. Yet, this will inevitably lead away from the actual problem, trying to cover all the (often not context-relevant) cases exposed through the declarative behaviour of the language. This 'overspecification' may in turn reduce the performance of the analysis tool significantly.

Basic syntax and semantics

In Alloy, every expression denotes a relation. Relations can be of any arity. Sets of atoms are represented by unary relations and scalars by singleton unary relations. Some of the available standard operators are the union + and intersection &. Depending on the type and arity of the given arguments, the operators will yield a (singleton) unary relation, binary or n-ary relation. The operators for comparing relations are = for equality and in for membership. Some available logical operators are negation !; conjunction &&; disjunction ||; and implication =>. Two of the available quantifiers are the universal quantifier all and the existential quantifier some. One of Alloy's strengths is its treatment and ease of navigation over complex relations. The available relational operators are the dot . for joining two relations; the product of two relations ->; the transpose of two relations ~; and the transitive ^ and reflexive transitive * closure operators.

In Alloy, two relations can be joined if the last atom of the first relation matches the first atom of the second relation. The resulting relation consists of the atoms of the first and second relation, leaving the matching atom out. When two relations a and b are joined in a.b, the resulting relation is obtained by taking every combination of a relation in a and relation in b and including their join. The product of two relations a and b is a->b and yields every combination of instances of a relation from a with the instances of a relation from b and concatenating them without dropping the intermediate atom. The treatment of scalars as singleton unary relations allows for the 'navigation' over fields of atoms. Given a scalar a, the expression a.b.c denotes the relation obtained by traversing from a over b to c.

If there are two basic types Principal and Role related through the relation has_member, then for a single role r the expression r.has_member would yield the relevant principals, while for a principal p the set of occupied roles can be obtained through the transpose p.~has_member. A policy object may be assigned to roles through the subject relation, and so the policy objects a principal holds through his roles can be obtained through the expression p.~has_member.~subject.

So, parts of the conceptual model (figure 1) can be represented in the form of the following signatures:

```
disj sig Object{}

disj sig Principal extends Object{
     has_active: set Role}

disj sig Role extends Object{
     has_member: set Principal,
     role_hierarchy: set Role,
     exclusive: set Role}
...
```

It can be seen how signatures are extended (extends) from the general object signature, a viewpoint similar to the object model of [2]. The disj keyword ensures that there are no overlaps between atoms. Fields such as has_member: set Principal describe relations between the relative signature (Role) and other signatures (Principal) with the keyword set indicating that several principals may be a member of a role.

This basic model can now be compiled and an initial analysis would reveal several problems that could occur, such as cycles in the role hierarchy. Such unintended properties need to be constrained by specifying invariants in the form of Alloy facts. Here, a role r must not be in the set of roles obtained by traversing downwards the role hierarchy.

```
fact {all r: Role| r !in r.^role_hierarchy}
```

Having specified the necessary signatures and facts, the next step is to define a set of functions to analyse the behaviour of objects. Such a function could be the assignment of a principal to a role, where s stands for the initial and s′ for the after state.

```
fun prin_role_assign (disj s,s′ : State,
prin : Principal, r : Role) {...}
```

However, Alloy does not provide a built-in notion of state. Instead, it allows the specification of states in the form of signatures, a technique called *objectification of state* [3].

Enabling dynamic analysis

Time is modelled as a sequence of states in the signature State_Sequence (1.). Such a sequence is defined by a first and a last state (2.), ordered through the relation next (3.). The !->! part of the relation says that this is a one to one relation, making the sequence a total order. Two explicit constraints are specified as part of the State_Sequence signature. Within a sequence, all states must be reachable from the first state (6.). Secondly, all states apart from the last (7.) must be related via one or more operations such as adding or removing a principal from a role (8.-12.).

```
1.   sig State_Sequence {
2.    disj first, last : State,
3.    next: (State-last) !->! (State-first)
4.   }
5.  {
6.    all s : State | s in first.*next
7.    all states : State - last |
8.    some s=states | some s'=states.next |
9.   (some prin:Principal | some r:Role|
10.    prin_role_assign (s, s', prin, r) ||
11.    prin_role_remove (s, s', prin, r)
12.  )
13. }
```

Using the constraint analyser, we can obtain a possible model of the specification where `Principal_2` and `Role_0` are not related in the initial state `State_0` but after the execution of the assignment operation in state `State_1`. Figure 3 is composed of parts of the analyser's output to reflect the state changes.

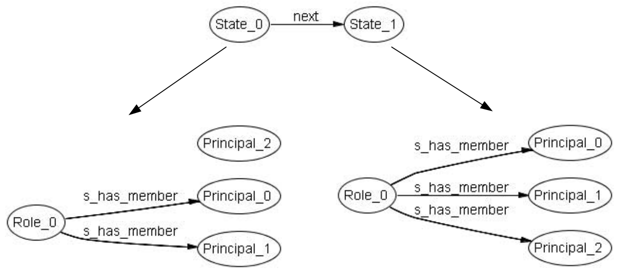

Figure 3: Assigning principals and roles

In order for each state to maintain information about its objects and their relations separately, the specification needs to be changed. For example, the `Role` signature with the `has_member` relation

```
disj sig Role extends Object{
    has_member: set Principal,
    ...}
```

has to be redefined in terms of a state signature

```
sig State {
    s_has_member: Role -> Principal,
    ...}
```

such that the previous binary relation is now part of a new ternary relation `State->Role->Principal`, thus allowing to maintain information about any changes between states.

Maintaining a history

One additionally required property for the later specification of control principles is that of a history mechanism to record activities such as that of a principal accessing an object or delegating policy objects or roles. So for a possible function

```
fun access (disj s: State, r: Role,
 prin: Principal, obj: Object) {…}
```

which specifies object access of a principal using some role, the relation

```
access_hist: Object->Principal->Role
```

shows how the access of a principal using a role and its associated policies on an object is recorded. There are several solutions of how to maintain a history. In this context history is maintained by using a `History` signature:

```
sig History{
 access_hist: Object->Principal->Role
 delegate_hist: Principal->Principal->Role}
```

This information is integrated into the state signature such that a state now has a reference to at most one specific history instance showing what happened for that specific state.

```
sig State{
  ...
  s_history: option History
  ...}
```

It is required that any two states maintain a distinct history in terms of the actual object. Otherwise, it might happen that if the same operation is performed twice, the involved states refer to the same object, leading to a loss of information. This, however, does not forbid two history objects to maintain the same history. The history mechanism avoids redundant information as, for example, a history in form of a log with all accesses so far for a state, would hold. This also allows the observation of what caused a state transition. Since states are part of an ordered sequence of states, any information can be gathered by traversing that sequence from the first to the current state.

3 Specifying Control Principles

Our discussion of control principles is based upon the distinction between authorisation and obligation polices, introduced in section 2.1.

3.1 Expressing separation controls

The separation of duties is probably the best understood control principle at present, as shown by the variety of existing work specifically in the areas of role-based access control and distributed systems management [7], [8], [9], yet no work seems to address the separation of duties within the context of other control principles. In its most general definition, the separation of duties can be best described as a means of preventing (in)advertent error and fraud through the general or context-dependent limitation of a principal's authority.

The discussion on separation of duties gained impetus with the advent of role-based systems. However, in most cases the term seems to be limited to authorisations, specifically because role-based standards such as RBAC96 do not define duty or responsibility but only authority in terms of assigned permissions [1]. Strictly speaking, none of the existing work on this principle addresses the separation of duties, but the separation of authority, and lacks a concept for expressing duty or responsibility. Even frameworks distinguishing between obligations and authorisations [2] actually only discuss how to separate authority and neglect the concept of obligation when demonstrating how to enforce separation of duties.

In contrast, the model presented here provides the notion of separation of both authorisations and obligations. In the following the implementation of some selected separation controls is outlined.

Simple static and dynamic separation controls

The static and dynamic separation of duties are enforced using the notion of exclusive roles [7]. To preserve simple static separation no pair of exclusive roles may be assigned to a common principal. In this case, the function takes a state, two disjoint roles and a role graph as its input. If the two roles are in the graph's conflict set then the intersection (&) of the sets of principals assigned to the roles must be empty (no) in that state.

```
fun ssod (s:State,r1,r2:Role,rg:RoleGraph){
 r1->r2 in rg.conflict =>
  no ((r1.(s.s_has_member) &
      (r1.(s.s_has_member))}
```

In a similar manner, a simple dynamic separation is defined, allowing for the assignment of exclusive roles to a common principal but preventing the simultaneous activation through the has_active relation.

```
fun dsod (s:State,r1,r2:Role,rg:RoleGraph){
 r1->r2 in rg.conflict =>
  no ((s.s_has_active).r1 &
      (s.s_has_active).r2)}
```

A simple form of analysis is to assert that if a state preserves static separation of authority it also preserves dynamic separation as no counterexample is generated [7].

```
assert ssod_implies_dsod {all s:State |
                          all r1,r2:Role|
fun ssod(s,r1,r2,rg)=>fun dsod(s,r1,r2,rg)}
```

Object–based separation controls

Object-based separation means that if a principal is in possession of any two conflicting roles then he must not access the same object in both of them. This can be expressed in the obj_sod constraint where the function

```
fun hist_set () : set State.s_history {
 result = {...}}
```

returns the history over all states so that using the relevant field (here access_hist) the needed information is retrieved.

```
fun obj_sod(){all s:State|
              all prin:Principal|
              all disj r1,r2: Role|
              all o: Object |
              all rg: RoleGraph|
(r1->r2) in rg.conflict  &&
(o->prin->r1) in hist_set().access_hist =>
(o->prin->r2) !in hist_set().access_hist
}
```

Further separation controls

In a similar manner, we have specified further controls such as operational or history-based separation in the context of this framework. Additionally, we have demonstrated how to model the degree of shared authorisations as suggested in [7]. It is worth noticing that all separation controls have been specified as Alloy functions instead of universal facts. This allows for their composition, e.g. static separation with partially shared authorisations, and later assertion of this composition.

3.2 Delegation and revocation

The delegation and subsequent revocation as a mechanism to decentralise has been addressed in a multitude of work at varying levels of granularity and differing contexts, e.g. [10], [11], [12].

However, it often seems to be the case that the mechanism as such is not really understood in terms of its organisational intent and so the distinction between administrative and user-based delegation activities as perceived in, for example [13], [14], [15], may seem artificial. After all, the administrator is also a user of some system. Therefore, if delegation were looked at from a technical perspective, i.e. some sort of object or variable assignment, then this distinction would be clearly obsolete. However, it is their intent that is distinct. The administrator delegates because he follows a set of procedures. The 'business user' delegates because he has some sort of business goal he wants to accomplish. In fact, an administrator is a user and his business is to administer. Another difference is that an administrator does not usually hold the object to be delegated but only has some administrative authority over it, while in the case of a user-based delegation the object must be held by the delegating principal.

Additionally, the organisational motivation behind delegation activities must be considered. The various activities that are performed within an organisation can be categorised according to their degree of similarity, regularity and repeatability. So the higher this degree the more can these activities be regulated. In this case, subjects are released from making decisions on their own and merely execute what is defined in organisational policies. The lower this degree, the more a principal has to make individual decisions that can not be regulated by policies. In this case delegation is a necessary means to cover situations for which there is no defined procedure. If delegation activities occur frequently or principals delegate some object indefinitely, then this indicates that the current organisational structure and procedures do not reflect the goals and tasks of the principal. The initially temporary delegation must now become part of the regular administrative activities shaping the formal organisational structure. Therefore, it can be argued that on the one hand delegation is a key mechanism for achieving structure, while on the other hand it is constrained by structure.

The following sections will discuss how some basic delegation properties can be expressed in Alloy, focusing on the difference between delegating authorisation and obligation policy objects.

Basic delegation properties

What would be the properties of a basic delegation function in this context? The most general definition is that before the delegation the delegating principal is in some way related to the object to be delegated, while after the delegation the receiving principal is related to the object. This can be expressed as follows in terms of Alloy, where the function `assigned_policy()` returns true if a policy object is assigned either directly or over a role to a principal in some state.

```
fun delegate (s,s':State| p1,p2:Principal|
              pol:PolicyObject) {
 assigned_policy (s, pol, p1) &&
 assigned_policy (s',pol, p2)}
```

Analysing the behaviour of this function it can be observed that with respect to the delegating principal, the object may or may not remain related to him, while the receiving principal might have been related to the object before the delegation. These observations are relevant to the distinction between authorisation and obligation policy objects. In the case of delegating authorisations, the delegated object may or may not remain related to the delegating principal. For example, if a principal needs additional help with a task due to his limited resources (such as time), he may transfer the needed authority to some other principal to assist him, at the same time retaining his authority. In the case of delegating an obligation, the delegating principal must not continue to hold the obligation, as it must always be clear who exactly holds an obligation at a given moment. Instead, a new obligation called a "Review" must be introduced to hold him to account for his delegation (Section 3.3). An extended delegation operation needs to cater for the delegation of authorisations and obligations in the manner described above, but there is no space to discuss this any further in this context. See [16] for a detailed discussion.

Basic revocation properties

Revocation properties such as propagation, dominance and resilience are discussed in [12]. The history of delegation activities that is preserved in our framework allows for the revocation of policy objects to follow these properties. Space does not permit a more detailed discussion and the following sections will focus on the more novel idea of review policies for delegated obligations.

3.3 Expressing review controls

It is necessary to hold to account persons who delegate obligations, because if not, obligations might be delegated without any assurance that they will be discharged. The activity of review describes a post-hoc control that aims at controlling delegated obligations.

In order for principals to be able to give an account of the obligation that they have delegated, they must review the evidence generated by the discharge of the obligation. This is done by creating a review policy corresponding to the delegated obligation and its evidence. This review is the result of specifically delegating an obligation and is a specific type of obligation itself (Figure 1). In order to be reviewed, an obligation must have evidence associated with it, in the form of a set of application dependent criteria that must be met, e.g. some log file that is to be produced. The actions defined for an obligation must correspond to this evidence, in order that it can be produced after discharging an obligation. The review policy created for a delegated obligation then defines review actions by which the generated evidence is reviewed. A consequence of our approach is that obligations can not be delegated without the prior definition of the evidence of their discharge. A necessary precondition we have to enforce is that the relation `reviewed_by: Evidence -> Review_Action` has been instantiated. The appropriate value of `Evidence` for a `Review_Action` is application dependent.

A model of a chain of delegated obligations can be seen in figure 4. This has been created through an Alloy specification following the proposed framework. Here `PolicyObject_0` is the obligation that has initially been subject to delegation, as a result of which the review `PolicyObject_1` was created. This has then been delegated as well, as it can be seen in the created review `PolicyObject_2`.

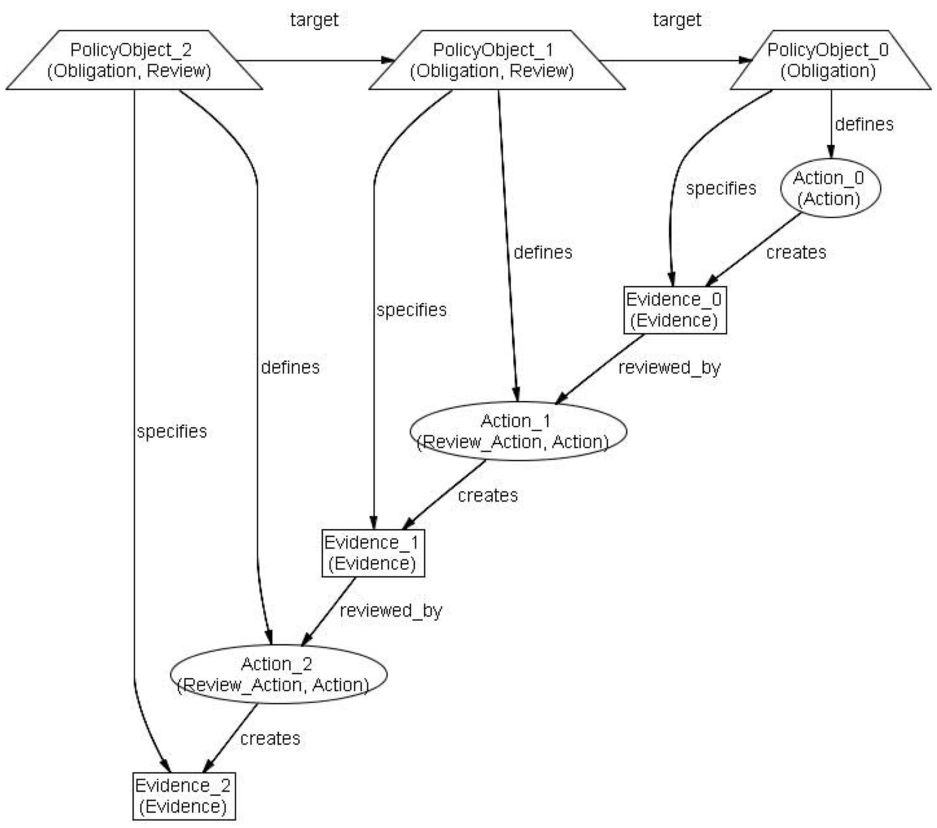

Figure 4: Delegated Obligation Chain

3.4 Expressing supervision controls

Delegation is one main means of achieving organisational structure. Two types of delegation have been identified in this context: delegation as an administrative mechanism to permanently create structure; and delegation as a temporary user-based mechanism to cope with situations not covered by the present structure and procedures. The latter has been outlined in the previous section. This section focuses on the former by discussing the more general supervision obligation of a principal.

Supervision is the general review obligation of a principal in a position to ensure that his subordinates carry out their assigned obligations. Accordingly, a supervision is defined as a binary `supervises` relationship between two positions. This `supervises` relationship, at its simplest, carries with it a set of obligations on the supervisor: To review all the obligations held by the subordinate.

The concept of evidence allows us to treat the obligations of a supervisor in the same manner as other review obligations. If the subordinate provides the evidence specified in his obligations, the supervisor can review it in order to generate the evidence that he in turn discharged his supervision obligation. This evidence might in turn be reviewed by his next superior etc., up the supervision management chain.

The inevitable question of the end of such a supervision chain is dependent on the context and structure of the organisation, but it is clear that there must always be an entity (e.g. Board of Directors) that need not generate evidence for any further review.

4 Analysing Control Principles

This section sketches an approach to control principle analysis with a focus on the types of analysis and the relationships between control principles. Technically, this was supported by using the state sequence approach (see section 2.3). So, after defining an initial and final state as well as possible state changing operations, the constraint analyser could be instructed to look for possible models within these given constraints.

The two types of control principle analysis that can be performed in this framework are:

- Control Principle Analysis
- Transition Analysis

Control principle analysis is the analysis of single or composed controls. Transition Analysis refers to the analysis of the behaviour of a system over sequences of states with respect to some defined properties and constraints. This is in line with the package structure discussed in section 2.2.

It has already been outlined how composed separation controls can be analysed (Section 3.1), and the following two sections will outline some more examples of static and dynamic composition.

4.1 Static analysis of composed controls

There is a strong correlation between separation and delegation controls. This can be best summarised by separation controls requiring or inhibiting delegation.

For example, a dual control might require that a principal must not perform an obligation on his own. Instead it must be either cross-checked or parts of it be separately performed by some other principal. In order to fulfil this obligation, the principal might have to delegate one of his roles or its assigned policies to this other principal to satisfy the dual control.

Another example is the case of a principal intending to delegate a role or some of its policies. This might violate the separation controls with respect to the receiving principal, e.g. if the object to be delegated is a role declared as exclusive to some role already held by the recipient.

4.2 Dynamic analysis of composed controls

There is also an example of delegation activities undermining separation controls that was discovered through Alloy's output. This considers the object-based separation of duty. Object-based separation says that a principal must not access the same object through two exclusive roles. The way this control had been initially specified by us allowed for the following sequence.

Two exclusive roles `r1`, `r2` are available to a principal `p1` in the initial state `S1`. Consider the state sequence:

```
S1: access(obj, p1, r1)
S2: delegate(r1, p1, p2)
S3: access(obj, p1, r2)
S4: revoke(r1, p1, p2)
```

The following can be informally observed. The principal `p1` accesses the object `obj` through his role `r1`. In the next state `S2`, principal `p1` delegates his role `r1` `to` principal `p2`. It is assumed that such a delegation leads to the temporary loss of that role. In the following state `S3`, `p1` accesses the same object through his remaining role `r2`, an action that might not have been intended. This is because in state `S3` the only role `p1` holds is `r2`. Depending on how the history of delegation actions is evaluated within the separation property, he will be granted access to the object and can revoke his delegated role in `S4`, effectively having accessed the object `obj` through both exclusive roles. Only through careful specification of the separation control and evaluation of the history of delegation activities for a subject can such scenarios be avoided.

4.3 Avoidance versus detection

It is a valid argument to claim that the violations and conflicts that were described in the previous section can be avoided by simple precedence rules, e.g. static separation must always be maintained. This is of course true, but the consequence of this may be that the organisation's business is obstructed by excessive control.

The alternative approach, which we advocate as a general approach, is setting static organisational controls at a level that may allow violations, and supplementing them with dynamic controls and post hoc reviews to maintain the required degree of control in the organisation.

5 Summary and Conclusion

This paper has presented a framework for the definition and analysis of organisational control principles using the Alloy specification language and its supporting tools. The underlying conceptual model integrates concepts of existing role- and policy-based distributed system management models using an object extension approach. The key aspects and novelties include:

- A discussion of the origins of control principles in the area of organisation theory, justifying and validating the components of this framework.
- A demonstration of the suitability of the Alloy language for specification and analysis of control principles.
- A specification architecture supporting static and dynamic analysis.
- A notion of relating roles with obligation and authorisation policy objects.
- A discussion on the basic principles and organisational motivations underlying the general delegation of system objects.
- An approach to delegating obligations and introduction of the concepts of evidence, review and supervision to control such delegation activities.
- The possibility to analyse controls on their own and in combination. This led to the discovery of previously unknown relationships, specifically between separation and delegation controls.

This may now be used to model and analyse aspects of real world organisations and we are currently investigating how a credit application process may be partially represented in our framework. Ideally, a workflow model would complement this. Other future work includes the further investigation of review and supervision controls. In particular it needs to be assessed what the effects of the actual discharge of a delegated obligation on the participants in the delegated obligation chain are.

It must be clear that any model can only abstract some of the complex characteristics of organisations and must leave many other questions unanswered. We nevertheless believe that this framework is a first step into integrating existing and new organisational control principles, providing security practitioners and researchers alike with a fresh perspective on security and control in organisations.

References

1. Sandhu, R., et al., *Role-based access control models.* IEEE Computer, 1996. 29(2): p. 38-47.
2. Damianou, N., et al. *The Ponder Policy Specification Language*. in *Policies for Distributed Systems and Networks*. 2001. Bristol: Springer Lecture Notes in Computer Science.
3. Jackson, D. *A Micromodularity Mechanism*. in *8th Joint Software Engineering Conference*. 2001. Vienna, Austria.
4. Merton, R., *"Bureacratic structure and personality"*, in *Reader in Bureaucracy*, R. Merton, et al., Editors. 1952, Free Press: New York. p. 361-71.
5. Moeller, R., *Changing definitions of Internal Control and Information Systems Integrity*, in *Integrity and Internal Control in Information Systems*. 1997, Chapman & Hall. p. 255-272.
6. Urwick, L., *Notes on the Theory of Organization*. 1952: American Management Association.
7. Kuhn, R. *Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems*. in *Proceedings of the second ACM workshop on Role-based access control*. 1997.
8. Simon, R. and M. Zurko. *Separation of Duty in Role-Based Environments*. in *Computer Security Foundations Workshop X*. 1997. Rockport, Massachusetts.
9. Gligor, V., S. Gavrila, and D. Ferraiolo. *On the Formal Definition of Separation-of-Duty Policies and their Composition*. in *IEEE Symposium on Security and Privacy*. 1998. Oakland, CA.
10. Harrison, M., W. Ruzzo, and J. Ullman, *Protection in Operating Systems.* Communications of the ACM, 1976. 19(8): p. 461-471.
11. Sloman, M. and J. Moffett, eds. *Delegation of Authority*. Integrated Network Management II, ed. I. Krishnan, Zimmer, W. 1991. 595-606.
12. Hagstrom, A., et al. *Revocations - A Categorization*. in *Computer Security Foundations Workshop*. 2001: IEEE Press.
13. Zhang, L., G. Ahn, and C. B. *A Rule-based Framework for Role-Based Delegation*. in *ACM SACMAT*. 2001. Chantilly, VA, USA.
14. Sandhu, R., V. Bhamidipati, and Q. Munawer, *The ARBAC97 model for role-based administration of roles.* ACM Transactions. Inf. Syst. Security, 1999. 2(1): p. 105 - 135.
15. Barka, E. and R. Sandhu. *Framework for Role-Based Delegation Models*. in *Annual Computer Security Applications Conference*. 2000. New Orleans.
16. Schaad, A. and J. Moffett. *Delegation of Obligations*. in *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*. 2002. Monterey, CA.

Acknowledgements

The UK Engineering and Physical Sciences Research Council has provided funding for Andreas Schaad under EPSRC Scholarship number 99311141.