IEEE Policies for Distributed Systems and Networks (POLICY 2002), to be held June 5-7, 2002, Monterey, CA

Delegation of Obligations

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

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

Abstract

Obligation policies are one main means of exercising control within an organisation. They specify the actions that some subject has to perform. The authority over these actions needs to be specified in authorisation policies. Current policy notations provide us with the needed structure to represent authorisations and obligations as policy objects for distributed systems management. They support the delegation of authorisations but not of obligations. Yet, there is a strong relationship between the two policy types and the delegation of obligations needs to be supported as well, requiring the introduction of a new type of policy which we call a "review".

This paper investigates the general principles underlying the delegation of policy objects, putting specific emphasis on the delegation of obligations. The Alloy specification language is used to specify and illustrate these principles. The main issues that will be discussed are: the balance between authorisation and obligation policies; the source of obligations and reasons for their delegation; the need for review policies to help control the delegation of obligations.

1 Introduction

Organisations have to achieve control over their activities. They do this by specifying policies expressing these control requirements. Policies range from abstract high-level policies down to refined policies which are expressed as rules defining a choice in the behaviour of the organisation. Obligations are one specific type of policy. They regulate which activities have to be performed by whom and when. In order to perform these activities and eventually discharge the obligation, the obligation holder needs sufficient authority which, in automated systems, is expressed in a set of access rights. Together with further information such as specific targets or constraints these form authorisation policies which represent a second specific type of policy.

The various activities that are performed within an organisation can be categorised according to their degree of similarity, regularity and repeatability. The higher this degree the more can these activities be regulated by policies. In this case, subjects are released from making decisions on their own and merely execute what is defined in those policies. The lower this degree, the more individual decisions have to be made which cannot be regulated by policies. The ultimate aim is to establish an organisation where a set of general policies regulates the core organisational activities at the same time leaving sufficient room for required individual decisions [1]. One mechanism which is employed to contribute to this aim is to allow for the delegation of policies between individuals. The Ponder policy framework [2] provides support for the delegation of authorisation policies but does not take the delegation of obligations into consideration. However, the delegation of obligations has been identified as a recurring phenomenon in distributed systems [3] that needs to be addressed.

2 Outline

Having given an initial introduction and motivation for the delegation of policies we introduce the Alloy specification language that will be used to model the delegation of obligations (Section 3). We show how the very basic properties of a delegation operation can be modeled and analysed in terms of Alloy and continue to further refine the initial specification to support the delegation of authorisation and obligation policy objects (Section 4). The important property of obligations requiring sufficient authority to be discharged is then specified (Section 5). We continue to concentrate on the source of obligations and identify a set of reasons for their delegation (Section 6). Review policies are introduced to control the delegation of obligations (Section 7). The paper finishes with a discussion on related work (Section 8); the properties of the new Alloy language that has been published recently (Section 9); and a summary and conclusion (Section 10).

3 The Alloy specification language

The Alloy language is designed for the specification of object models through graphical and textual structures [4]. It has a simple ASCII textual notation of which a subset can be expressed graphically. It is a state-based language and invariants constrain the relationships between objects. Alloy is supported by the Alloy Constraint Analyzer [5] which allows us to analyse specifications in order to detect over- and under-constraints. Alloy's syntax is based on the Z language [6] and integrates further concepts that are used in other object modeling notations [7].

Each state component is either a set, a binary relation, or an indexed relation. For sets, there are the usual set-theoretic operators (shown here in ASCII form):

`s + t`	union of s and t
`s & t`	intersection of s and t

There are no set constants in Alloy. Instead, we have:

`some s`	s is non-empty
`no s`	s is empty
`one s`	s has exactly one element

Elementary formulas in Alloy are made by comparing sets (which can be singletons):

`s = t`	equality: s and t have the same elements
`s in t`	subset: every element of s is an element of t

Alloy has standard quantifiers. If a variable x is part of a formula F we can write:

`all x:s|F`	F is true for every value of x in the set s
`some x:s|F`	F is true for some value of x in the set s

Standard logical operators are provided:

`&&`	and
`||`	or

An important operator in Alloy is the relational image '`.`' [4]. The expression `s.r` denotes the set of objects that the set `s` maps to in the relation `r`. This kind of expression is often called a "navigation expression" because we navigate from `s` along a relation `r`. Applying the image operator again yields a longer navigation. Suppose, for example, that we have the sets `Subject`, `Obligation`, and `Action` denoting the components of a simple managed system. The relation `holds` maps the sets of subjects to obligations. Then for a single subject `s1`, the relation `s1.holds` gives us the set of obligations that the subject holds. Introducing a further relation `requires`, which maps an obligation with required actions, we can now write `s1.holds.requires` to denote the set of all actions a subject requires to discharge his obligations.

Other relational operators include the transpose of a relation (`~`) and its transitive closure (`+`).

4 Delegation of policy objects

Policy objects represent policies in an automated system. Within frameworks such as the Ponder language two such objects are authorisation and obligation policies. They specify what subjects can and have to do.

One form of decentralising control in organisations is to allow for the delegation of these policy objects from one subject to another. Technically, delegation is the activity of creating a new relation between a subject and an existing policy object. We believe that there are some general principles underlying the delegation of the two types of policies. However, a distinction has to be made between the specific delegation properties of authorisation and obligation policies.

4.1 General delegation properties

The initial general model of policy objects that we will use for specifying and analysing delegation properties is as follows:

```
model policy_objects{
      domain{PolicyObject, Subject}
      state{
      holds : Subject -> PolicyObject}
}
```

The domain paragraph describes the type of objects we make use of. The state paragraph then describes the relations between the objects. Where it is not specified the cardinality of a relation is automatically assumed to be zero or more. Otherwise, the plus sign and the exclamation mark are used to indicate that, for example, a subject holds one or more (`+`) policy objects or that a policy object can only be held by exactly one (`!`) subject.

The delegation of a policy object is informally defined as an operation, where one subject *delegates* a policy that he holds to another subject, who holds the policy after the operation completes.

We now want to specify an operation (`op delegation_1`) describing the change of state for a delegation activity. As in other sequential specification languages, the before state is left unmarked while the after state is indicated by the primed symbol (`'`). The argument list of the operation defines the variables the operation is carried out upon.

This operation says that, before the operation, the delegating subject `subj1` must hold the policy object to be delegated (`pol_obj in subj1.holds`). It is silent about whether `subj1` still holds the policy object after the operation. However, after the operation `subj2` must hold the policy object (`pol_obj in subj2.holds'`).

```
op delegation_1(subj1,subj2:Subject!,
                pol_obj:PolicyObject!){
//Subjects are distinct
subj1 != subj2

//Maintain other relationships
all subjs : Subject – (subj1 + subj2) |
subjs.holds' = subjs.holds

all objs: PolicyObject – pol_obj |
objs.~holds' = objs.~holds

//Delegate policy object
pol_obj in subj1.holds
pol_obj in subj2.holds'
}
```

We can now use the analysis facilities of Alloy to find a model that satisfies our formulae within a defined search scope (e.g. for up to 2 subjects and 2 policy objects). Such a graphical representation helps us in detecting any under- or over-constraints or other undesired model properties. Based on the above specification, we prompt the Alloy constraint analyser to generate the following two models with respect to the execution of the delegation operation:

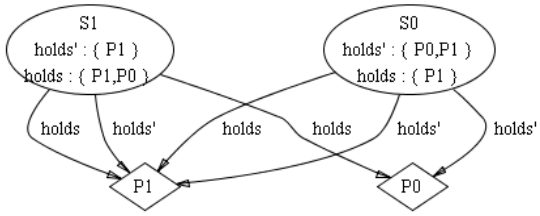

Figure 1: Basic Delegation Model 1

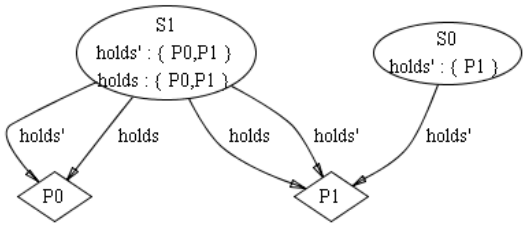

Figure 2: Basic Delegation Model 2

These examples have been generated arbitrarily by Alloy, which also determines the labeling (e.g. reversal of S0 and S1 between figures 1 and 2). Changing the search scope might result in different models. Only the constraints given in the object model and the operation define what a well-formed state should look like and how the before and after state of an operation should relate to each other.

It can be observed in figure 1 (and from further output given by the constraint analyser) that the policy object P0 was delegated from subject S1 to subject S0. S0 did not previously hold P0, while S1 did. After the delegation S0 now holds P0 and S1 does not anymore. Similarly, it can be seen in figure 2 that P1 was delegated from S1 to S0. The difference from the delegation in figure 1 is however, that the policy object remains with the delegating subject.

Our specification shows a basic structure underlying the delegation of policy objects. However, a general delegation model will not suit all types of policies. The reasons for this and the properties that have to be defined for the delegation of specific policy objects are discussed now.

4.2 Partitioning policy objects

We show in this section why a general delegation model is insufficient, because different constraints apply to the delegation of authorisations and obligations. Our initial specification has to be extended so that we can talk about authorisations and obligations as specific policy objects. This is a similar viewpoint to that expressed in the Ponder framework.

In order to treat authorisations and obligations separately, the Alloy mechanism of partitioning the `PolicyObject` domain into `Authorisation` and `Obligation` domains is used. Accordingly, the following formula is added to the state paragraph of our initial specification:

```
partition Authorisation,
          Obligation: PolicyObject
```

This expresses the fact that the sets of authorisation and obligation objects are mutually disjoint and that their union will result in the set of policy objects.

4.2.1 Delegating authorisations

When authority is delegated between two subjects the general intent of the delegating subject is to give the receiving subject the power to act on its behalf.

The delegating subject must already hold the authority object and also hold another piece of authority which allows him to initiate the delegation process. After the delegation took place the delegator still holds the authority object, while the receiving subject now also holds it. Subsequently, we allow multiple subjects to hold the same authorisation object. This could lead to a situation where a subject already holds an authorisation that it is supposed to receive in a delegation. This makes sense as delegated authority can also be revoked, however, the semantics to maintain a history of delegated authorisations is outside the scope of this paper.

Considering the direction of a delegation of authority we usually think in terms of a downwards delegation along a management chain. However, delegation can also

be directed up that chain. It is not necessarily the case that the authority of a subordinate is a subset of that of his superior. Often specific authority is held by subjects and they can delegate this to their superior. Similarly, subjects can delegate authority to other peer subjects along a horizontal scale.

4.2.2 Delegating obligations

When an obligation is delegated between two subjects the general intent of the delegating subject is to make the receiving subject perform a set of activities. The reasons for this vary and we discuss some examples in section 6.2.

In general, an obligation must be held by a single subject in order to ensure that tasks are only carried out once. Although there are some exceptions in real life, we are enforcing this constraint in this paper. This implies that after the delegation took place the delegating subject no longer holds the obligation policy object. We will later see that this gap needs to be filled by the creation of a new obligation (Section 7.1).

Similar to the delegation of authority we can observe that obligations are usually delegated downwards along a management chain but in certain cases (e.g. illness of an employee) an obligation might be delegated from a subordinate to his superior. Likewise, a horizontal delegation of an obligation can occur.

4.3 Extending the delegation operation

We now specify a delegation operation that specifies different results for authorisations and obligations; an authorisation policy object remains held by the delegating subject after the operation, whereas an obligation does not. The following Alloy operation captures this requirement:

```
op delegation_2 (subject1, subject2:
      Subject!,pol_obj: PolicyObject!){

//Maintain relationships
all policies : PolicyObject - pol_obj |
  policies.~holds' = policies.~holds

//Delegate Authority
pol_obj in Authority ->
 (pol_obj in subject1.holds &&
  pol_obj in subject1.holds' &&
  pol_obj in subject2.holds')

//Delegate Obligation
pol_obj in Obligation  -> (
  pol_obj in subject1.holds &&
  pol_obj !in subject1.holds' &&
  pol_obj in subject2.holds')
}
```

This operation expresses that depending on what kind of policy object is delegated, the constraints on the before and after state are different. In case of delegating authority (`pol_obj in Authority ->...`), the expression `pol_obj in subject1.holds` says that the policy object must be assigned to the delegating subject in the before state, while `pol_obj in subject1.holds'` expresses that this initial relationship is also maintained in the after state. In case of delegating an obligation (`pol_obj in Obligation ->...`), the delegating subject will not hold the policy object anymore as indicated by the exclamation mark in the expression `pol_obj !in subject1.holds'`.

The analysis of the specification illustrates the different behaviour of the operation with respect to a specific policy object. Figure 3 shows that after the delegation of an authorisation policy an extra relationship has been created, whereas in figure 4 the delegation of an obligation results in the transfer of the policy object.

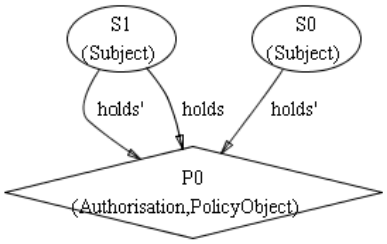

Figure 3: Delegation of an Authorisation

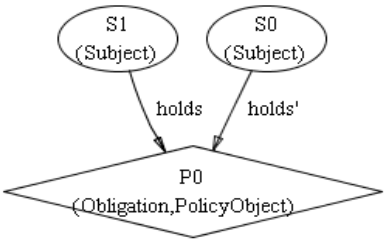

Figure 4: Delegation of an Obligation

5 Obligations require authority

Obligation policies need a set of corresponding authorisations such that the required actions can be performed and the obligation can be discharged [8]. More specifically, the action specified in an obligation for a subject and possibly target must match an action in an authorisation with the same parameters. This is an obvious but important relation between obligations and authorisations, specifically within systems that allow for the delegation of authorisations and obligations [9], [10].

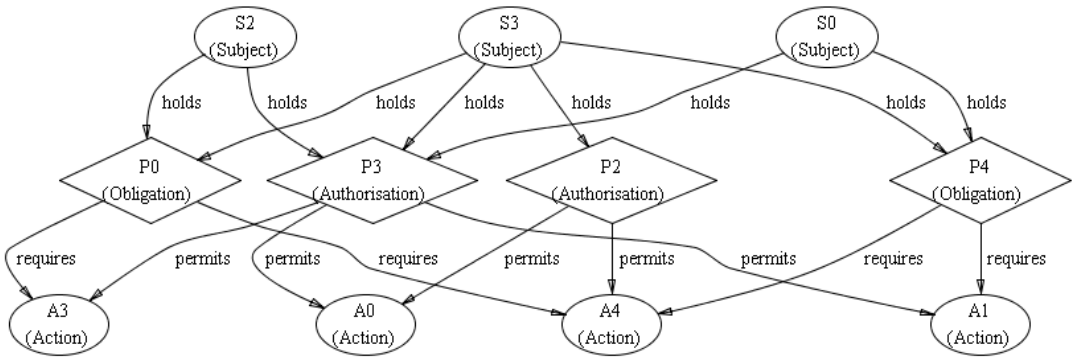

Figure 5: Unbalanced Model

If an authorisation is delegated from a subject `s1` to another subject `s2` without a corresponding obligation on `s2`'s side, then the principle of least privilege is violated.

If, on the other hand, an obligation is delegated by `s1` without the needed authorisations, then `s2` will not be able to discharge the obligation, subsequently reducing the operational efficiency of the system, but possibly even making `s2` liable for the failed discharge.

As a result, we have to specify a set of constraints on the relationship between an obligation, an authorisation, and their associated actions. These constraints aim at establishing the balance of an obligation and authorisation configuration set. We will now continue to use the Alloy specification language to express and analyse this requirement and its implications, within the so far established model that allows for the delegation of authorisations and obligations.

First we have to extend the initial model to include actions. We do this by adding the following two relations to our state paragraph:

```
requires: Obligation -> Action+
permits:  Authorisation -> Action+
```

These relations state that an authorisation consists of one or more actions and that an obligation requires one or more actions. If we now generated a model according to this so far unconstrained specification we can easily see the problems described earlier on.

Figure 5 shows that S2 will not be able to discharge his obligation P0 as one of the required actions A4 is not associated to any authorisation he holds. On the other hand S3 has more authority then he needs (A0) to discharge his obligations P4 and P0. There are now several ways to achieve the desired balance between obligations and authorisations, expressed in the following three invariants:

- Obligation-Centric
  We require that the set of required actions a subject holds through its obligations must be a subset of all the permitted actions which are part of the authorisations the subject holds. This can be expressed in the following invariant:

```
inv obligation_centric {
all s1: Subject |
 some s1.holds.requires ->
      s1.holds.requires in
      s1.holds.permits}
```

  While this invariant ensures that a subject can always discharge its obligations, it will still allow a subject to have more authority then it perhaps needs. We thus specify the next invariant.

- Authorisation-Centric
  We require that the set of permitted actions a subject holds through its authorisations must be a subset of all the required actions which are part of the obligations the subject holds. This can be expressed in the following invariant:

```
inv authorisation_centric{
all s1: Subject |
 some s1.holds.permits ->
      s1.holds.permits in
      s1.holds.requires}
```

  This invariant is just the dual to the previous invariant and will ensure that there is no non-required authority.

- Well-balanced
  We might want to additionally require the model to be well-balanced, such that for each obligation there exists exactly one matching authorisation.

```
inv well_balanced {
all o: Obligation |
 one a: Authorisation |
    o.requires = a.permits &&
    one (o.~holds & a.~holds)}
```

  For reasons of space, we just generate a model that satisfies all three constraints (Figure 6).

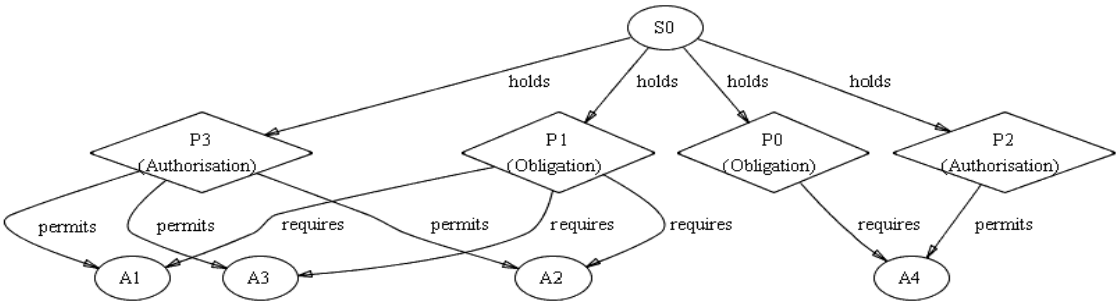

Figure 6: Well-balanced model

6 Organisational obligations

6.1 The source of obligations

Obligations are derived from an organisation's set of top-level goals and as such are driven by more general legal, moral and most importantly economical goals.

Their source is often identical to the source of authority [11], as both types of policy are usually specified together. Thus, high-level obligations are created by the stakeholders (e.g. shareholders) of an organisation and are initially placed upon the entity controlling the organisation (e.g. board of directors). From there obligations are refined and delegated down through the organisational hierarchy together with the required authority.

Most types of organisations have identified and implemented a set of core business processes to achieve their goals, and obligations and authority are usually expressed in the job descriptions of the human participants of a process. Even in the case of automated systems used to support the main business processes of an organisation, any obligations should trace back to human managers. For example, the obligation of the server to make a nightly backup is actually the obligation of the network administrator. Equally, the obligation of a network switch to provide quality of service (QoS) functions is actually part of the more general obligation of a human manager to fulfill his QoS contracts.

6.2 Delegating obligations

We saw that one reason for delegating obligations is the refinement of higher-level obligations into chunks that are manageable and can be eventually discharged by a subject. However, there are further organisational motives behind the delegation of an obligation. Some examples of such motives are:

1. Lack of resources
   A subject has not the resources sufficient to discharge an obligation it holds. Examples for such resources could be a lack of time, equipment, or missing domain membership

2. Competence
   The subject is not sufficiently competent to perform an activity. It might hold the obligation, but will have to delegate parts or all of it in order to discharge it.

3. Specialisation
   The subject might be sufficiently competent to discharge an obligation but it is more efficient to delegate parts or all of the obligation to subjects in specialist positions (e.g. discharge takes less time).

4. Organisational policies
   Specific organisational policies such as separation of duty or dual control rules require the delegation of an obligation. In the first case, the obligation might have to be delegated because the subject already holds another obligation, in the second case, the obligation might have to be delegated because it can only be discharged by two subjects.

These examples show that we can have either organisational (1-3) or policy-based (4) factors which cause the delegation of an obligation.

7 Organisational control through review

The continual creation, delegation and discharge of obligations causes unstable situations within an organisation. Unstable means that we are often uncertain about who currently holds an obligation, whether somebody has discharged his obligations, the effect of such a discharge and who has to make sure that some tasks are performed in the end.

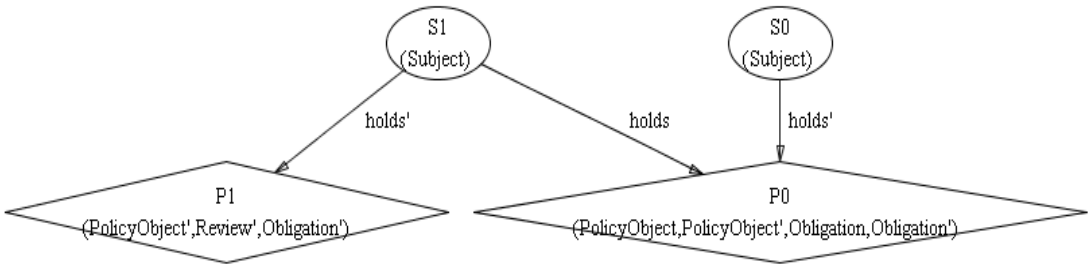

Figure 7: Creating a Review obligation

For this reason it is necessary to hold to account persons who delegate obligations. In order for them to be able to give an account of the obligation that they have delegated, they must review it. This is done by creating a review policy, described below, corresponding to the delegated obligation.

The activity of review describes a post-hoc control that aims at controlling delegated obligations. A review policy is created as the result of specifically delegating an obligation and is a specific type of obligation itself (Figure 8).

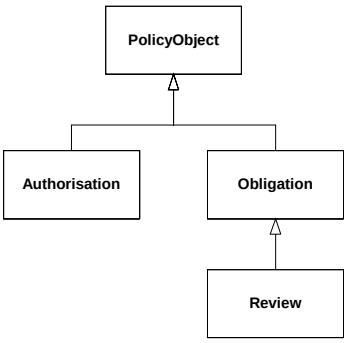

Figure 8: Policy Object Types

We can represent this fact within the so far established model by declaring a review to be a subset of obligation in the expression `Review:Obligation`. This is in line with the view expressed in the Ponder object meta-model and if required, any further specific obligation types found could be integrated by partitioning the model as shown in the previous section (4.2). However, note that Alloy does not talk about inheritance as such.

What is the relationship between the actions in an obligation policy that has been delegated, and the actions in the review policy that is created in its place? This is application-dependent; we assume in this context that there is a relation `reviewed_by: Action! -> Action!`, which defines the review action for each obligation action. We also need to define a relation `target: Review! -> Obligation!` to express the target of a review policy.

7.1 Expressing review policies in Alloy

We now look at the effects of introducing review policies in terms of our current delegation of obligation operations. We have seen how a policy object is generally delegated between two subjects and that there is a difference between delegating authorisations and obligations, namely that obligations must be assigned to a unique subject. When an obligation is delegated, the delegating subject loses its assignment to the obligation, but a new review obligation policy object is created and assigned to him.

This can be captured by the adding the following statements (bold) to our current delegation operation (Section 4.3). We declare one new review object to be created (`a_review: Review'!`) and assign this to the delegating subject after the delegation

```
op review_delegation_2
(subject1,subject2:Subject!,
 pol_obj: PolicyObject!,
 a_review: Review'!){

//review object doesn't initially exist
  no review1 & PolicyObject.

//Maintain relationships and
//Delegate Authority
  ...(see earlier specifications)

//Delegate Obligation
  pol_obj in Obligation  ->
  pol_obj in subject1.holds &&
  pol_obj in subject2.holds' &&
  pol_obj !in subject1.holds' &&
  a_review in subject1.holds'
  ...
}
```

The effects of this operation can now be observed in figure 7 and we can see how a new review obligation P1 was created for the delegating subject S1.

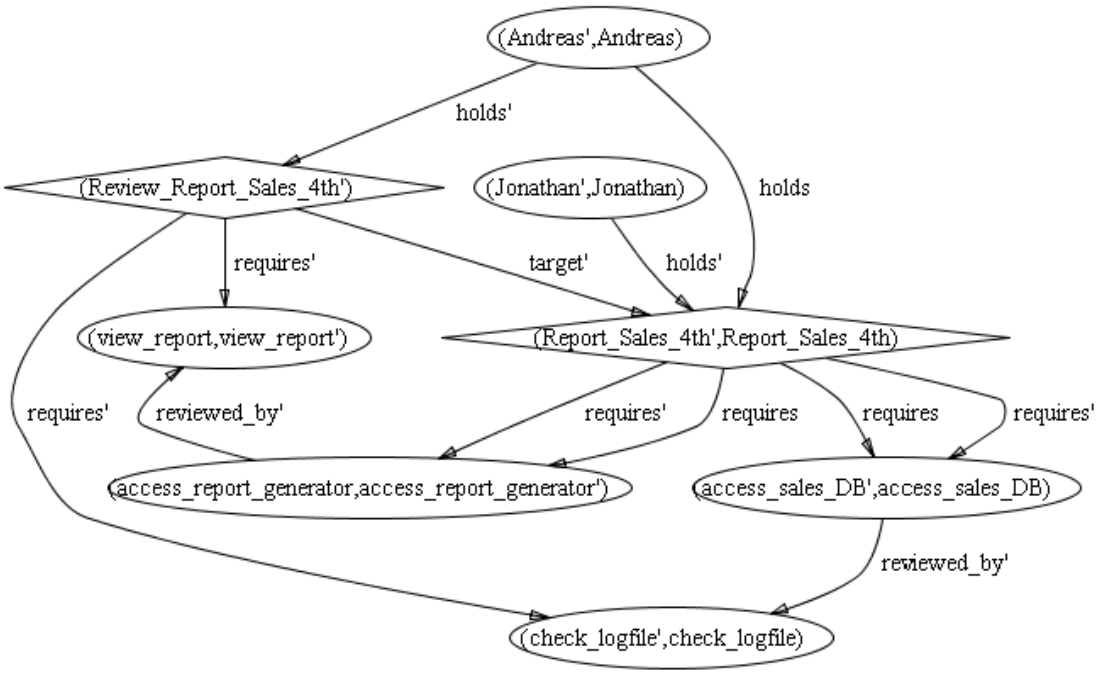

Figure 9: Example of a specific delegation

7.2 Review policies in automated systems

We said that a review policy is the obligation of a subject to investigate the state of affairs of an obligation it delegated to another subject. In other words, review is an obligation on an obligation where the required actions provide the application specific information on how to perform the review. Let us illustrate this in the following example, where we used Alloy to create a delegation between two specific subjects, at the same time supporting this operation through the declaration of Ponder-style obligation policy instances.

We can imagine an obligation for Andreas to prepare the quarterly sales report to look like:

```
Obligation Report_Sales_4th:

on          before 01/05/01;
held by     Andreas;
target      Sales_Database;
requires    access_report_generator and
            access_sales_DB;
```

This says that Andreas must prepare the report by performing the required actions on the sales database. Current business requires Andreas to delegate this task to Jonathan. In his role as the delegator, Andreas has to review that Jonathan carried this task out satisfactorily before the deadline. The actions that need to be carried out are application-dependent, and are defined in the `reviewed_by` relation. The first column represents the obligation actions and the second the review actions.

```
reviewed_by:

access_sales_DB         | check_logfile
access_report_generator | view_report
```

From this information a review obligation can be created:

```
Review Report Sales 4th:

on          before 01/05/01;
held by     Andreas;
target      Report_Sales_4th;
requires    check_logfile and
            view_report;
```

Andreas can now discharge his review obligation by looking at the database log file for access by Jonathan and by viewing the generated report file.

We can now observe in figure 9, how Andreas delegates his obligation to Jonathan, while at the same time a new review obligation with the name `Review_Report_Sales_4th'` is created for him as a result of this delegation. We can further observe how the two actions required by the obligation are related to the two review actions.

With respect to the authorisation policies and the balance between them and obligations, we can make the following observations for review policies. The creation of a review needs to be supported by the creation of a matching authorisation policy, allowing the reviewer to perform the demanded actions. This implies that an obligation cannot simply be delegated but that it must belong to a set of obligations for which the relevant review activities have been identified earlier.

Technically, the delegation of a review is performed just like the delegation of an ordinary obligation with the result that a new review policy is created to review the review of an obligation. We have not investigated the feasibility of this yet, but there are some obvious requirements such the absence of cycles in such a delegation chain.

8 Related work

8.1 Delegation of obligations in Ponder

Ponder [2], [12] is a declarative, object-oriented language for the specification of security and management policies in networks and distributed systems. Ponder policies relate to system objects and control the activities between them through authorisation; obligation; refrain; and delegation policies within a defined set of constraints. Additional constructs such as groups, roles, relationships and management structures further facilitate system management.

Ponder specifies that when the delegate() method is executed a separate authorisation policy is created with the grantee as the subject. Within the context of our Alloy specification we assumed that one authorisation can be held by multiple subjects (Section 4.2).

While Ponder explicitly supports the authority to delegate authorisation policies, it supports neither the obligation to delegate authorisation policies nor the delegation of obligations. Let us explain this in the following example with subject `A` delegating `print` (action) `report` (object) to subject `B`. This needs the following policies (using a very compressed version of the Ponder syntax), where the roman numbers represent the relevant entry in the matrix (Figure 10):

I. A policy which authorizes `A` (subject) to create an authorisation policy for `B` to `print report` (`B` is grantee). This is a Ponder delegation policy (`P0`) and must refer to an authorization policy `P1` (see II.) which exists when the delegate method is executed:

```
P0: deleg P1, A, B, report, print
```

II. An authorisation policy `P1` for `A` to `print report`:

```
P1: auth+ A, report, print
```

III. An obligation policy to perform the delegation of authority. There is no specific policy of this kind defined in Ponder. It would be similar to a `deleg` policy in referring to the pre-existing `auth+` policy `P1`.

IV. A policy which authorizes `A` (subject) to create an obligation policy for `B` to `print report` (`B` is grantee). This is not defined by Ponder and must refer to an obligation policy `P2` (see V. below) which exists when the delegate method is executed on it.

V. An obligation policy `P2` for `A` to `print report`:

```
P2: oblig A, report, print
```

VI. An obligation policy to perform the delegation of obligation. There is no specific policy of this kind defined in Ponder. It would be similar to a `deleg` policy in referring to the pre-existing `oblig` policy `P2`.

If any of the above policies are missing, delegation will not take place. The extent to which these are supported by Ponder is summarized in figure 10. We have not investigated the practicality of writing "raw" Ponder and OCL to achieve the effect of items III, IV and VI

Delegate Authority		Delegate Obligation	
I	`deleg`	IV	not explicitly supported
II	`auth+`	V	`oblig`
III	not explicitly supported	VI	not explicitly supported

Figure 10: Ponder Coverage Matrix

While Ponder does not explicitly support the authority to delegate obligations, or the obligation to delegate authority or obligations, its flexible object-oriented design and policy object meta-model could support these, and the further integration of review controls.

8.2 Further related work

The delegation of obligations within distributed systems is further discussed in [3]. A distinction is made between the transfer of an obligation; sharing and splitting

an obligation; and outsourcing an obligation. Only the outsourcing of an obligation can be considered as a delegation within our context while the other three activities refer to general obligation management that do not create a chain of delegated obligations. We specifically ruled out the sharing of an obligation. The splitting of obligations could be represented in our approach by a re-assignment of the relevant actions.

The delegation of obligations and responsibility has also been described in the context of the ORDIT methodology [13, 14]. A distinction is made between consequential and causal responsibility. Consequential responsibility is a ternary relationship between two agents and a state of affairs, and causal responsibility is a binary relation between an agent and an event or a state of affairs. While an agent is responsible for something, an obligation expresses that he has to (not) do something. A binding between an obligation and a responsibility is created through a state of affairs they are associated with. The obligation maintains or changes the state of affairs for which the responsibility is held. Obligations can be delegated and the new obligation holder becomes responsible to the original obligation holder for discharging the obligation. The view that there is always an ultimately responsible principal is consistent with the observations made in [9] and the responsibility principle established in [10].

9 The new Alloy language specification

Shortly after this paper had been written, a completely revised version of the Alloy language has been presented [15]. This addresses most of the deficiencies and drawbacks of the language as it was presented in this context and discussed elsewhere [16]. Old Alloy specifications cannot be directly analysed by the new constraint analyzer anymore, however, only minor changes are needed to translate old Alloy specifications into the current version. The most significant changes in the new Alloy language and its analysis facilities include [17]:

- Structuring mechanism for incremental extension.
- Support for modularisation of specifications.
- No explicitly built-in notion of state invariants and operations; instead the possibility of sequential composition of operations and modeling arbitrary sequences of states.
- Support for libraries of polymorphic datatypes such as lists, sequences or trees.
- Support for n-ary relations.
- Support for cardinality constraints and basic integer operations

We have already re-specified the results of this paper and further applied the sequential composition of states technique introduced in [17]. This will allow us in our planned future work to investigate the effects of generating chains of delegated obligations and their discharge within a defined sequence of states.

10 Summary and conclusion

In the course of this paper we have discussed the general principles underlying the delegation of policy objects with a special emphasis on the delegation of obligations and authorisations. Using the Alloy lightweight formalism helped to enhance the clarity of this work. We observed that obligations require sufficient authority and presented alternative constraints to achieve the appropriate balance between them. We further showed that the source of an obligation is identical to that of authority and that both policies are refined and propagate together in an organisation. We also observed that the delegation of obligations can result in uncertainty about whether they are discharged. The organisational control principle of review is a means of providing accountability for obligations. We therefore introduced review polices which oblige subjects to perform appropriate review actions for delegated obligations.

We have discussed the relationship of the Ponder policy language to our work, and showed that it does not provide explicit support for the delegation of obligations, or for the obligation to delegate authorisation or obligation policies. However, its flexible object-oriented design and policy object meta-model could support these, and the further integration of review controls.

Our future work will concentrate on the extension of the work presented here, specifically taking into account further control principles such as Supervision and meta-policies like the Separation of Duties. By integrating these in a common framework we will be able to investigate possible relationships between control principles.

11 Acknowledgements

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

Daniel Jackson and Mandana Vaziri at the Software Design Group, MIT have given valuable comments considering technical issues of the Alloy language.

References

[1] Woehe, G. and U. Doering, Einfuehrung in die Allgemeine Betriebswirtschaftslehre 1996, Muenchen: Verlag Franz Vahlen

.

[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] Cole, J., et al. Author Obliged to Submit Paper before 4th of July: Policies in an Enterprise Specification. in Policies for Distributed Systems and Networks. 2001. Bristol, UK: Springer Lecture Notes.

[4] Jackson, D., Alloy: A Leightweight Object Modelling Notation. 2000, MIT Laboratory for Computer Science: Cambridge, MA.

[5] Jackson, D., I. Schechter, and I. Shlyakhter. Alcoa: the Alloy Constraint Analyzer. in Proc. International Conference on Software Engineering. 2000. Limerick, Ireland.

[6] Spivey, M., The Z Notation: A Reference Manual. Prentice Hall International Series in Computer Science. 1989: Prentice Hall.

[7] Warmer, J. and A. Kleppe, The Object Constraint Language: Precise modeling with UML. 1998: Addison Wesley.

[8] Moffett, J. and M. Sloman, Policy Conflict Analysis in Distributed System Management. Ablex Publishing Journal of Organisational Computing, 1994. 4(1): p. 1-22.

[9] Mullins, L., Management and Organisational Behaviour. 1993, London: Pitmans Publishing.

[10] Urwick, L., Notes on the Theory of Organization. 1952: American Management Association.

[11] Moffett, J. and M. Sloman, The Source of Authority for Commercial Access Control. IEEE Computer, 1988(February): p. 59-69.

[12] Damianou, N., et al., Ponder: A Language for Specifying Security and Management Policies for Distributed Systems - The Language Specification. 2000, Imperial College: London.

[13] Dobson, J. and J. McDermid. A Framework for Expressing Models of Security Policy. in IEEE Symposium on Security and Privacy. 1989. Oakland, CA.

[14] Dobson, J. New Security Paradigms: What Other Concepts Do We Need as Well? in 1st New Security Paradigms Workshop. 1993. Little Compton, Rhode Island: IEEE Press.

[15] Jackson, D., Micromodels of Software: Lightweight Modelling and Analysis with Alloy. 2001/2002, Software Design Group, MIT Lab for Computer Science.

[16] Mikhailov, L. and M. Butler. Combining B and Alloy. in ZB 2002: Formal Specification and Development in Z and B. 2002. Grenoble, France.

[17] Jackson, D. A Micromodularity Mechanism. in 8th Joint Software Engineering Conference. 2001. Vienna, Austria.