Revocation of Obligation and Authorisation Policy Objects

Andreas Schaad

SAP Research, 805, Av. Dr. Maurice Donat,
06250 Mougins, France
andreas.schaad@sap.com

Abstract. In [Schaad and Moffett, 2002] we have presented our initial investigations into the delegation of obligations and the concept of review as one kind of organisational principle to control such delegation activities. This initial work led us to a more detailed and refined analysis of organisational controls [Schaad, 2003], [Schaad and Moffett, 2004] with a particular emphasis on the notion of general and specific obligations [Schaad, 2004]. In particular, this distinction allowed us to formally capture how a principal may be related to an obligation; how obligations relate to roles; and how the delegation of specific and general obligations may be controlled through the concepts of review and supervision. This paper complements the delegation of obligation and authorisation policy objects by discussing their revocation, based on the revocation schemes suggested in [Hagstrom et al., 2001]. In particular, we will investigate how delegated general and specific obligations can be revoked and what effect the presence of roles has on the revocation process. We use the Alloy language and its automated analysis facilities [Jackson, 2001] to formally support our discussion.

1 Introduction

Organisational control principles, such as those expressed in the separation of duties, delegation of obligations, supervision and review, support the main business goals and activities of an organisation. A framework has been presented in [Schaad, 2003] where organisational control principles can be formally expressed and analysed using the Alloy specification language and its constraint analysis tools [Jackson, 2001]. Specifically the delegation of obligations and arising review obligations has initially been treated in [Schaad and Moffett, 2002] and later expanded in [Schaad, 2004]. The delegation of policy objects must be complemented by their revocation. However, specifying revocation controls may be very complex as, for example, demonstrated in the work of [Griffiths and Wade, 1976], [Jonscher, 1998] or [Bertino et al., 1997], addressing revocation of permissions in the context of operating and database systems. A more general framework for revocation has only been proposed recently [Hagstrom et al., 2001]. This is, however, limited to the revocation of permissions directly assigned to a principal and does not include a notion of roles. Our paper explores how this revocation framework can be applied for the revocation of obligation and authorisation policy objects in the context of our control principle model. Specifically our distinction between general and specific obligations requires a more detailed discussion of possible revocation schemes.

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 28–39, 2005.

The rest of this paper is structured as follows. Section 2 provides a summary of the core static and dynamic concepts of our control principle model, a more detailed formal discussion of which is provided in [Schaad, 2003]. Section 3 will then look at the revocation of policy objects against the dimensions of resilience, propagation and dominance, in particular focusing on the revocation of obligations. Section 4 summarises and concludes this paper.

2 Definition of Policy Objects in the Control Principle Model

Within our control principle model [Schaad, 2003], policy objects are either authorizations or obligations, similar as those defined in Ponder [Damianou et al., 2001]. Principals, or the roles of which principals are a member of, may be subject to these policy objects. In other words, a principal is related to a set of policy objects over the roles he holds or on the basis of a direct assignment. The target of a policy object defines the objects against which the actions of the policy are executed.

Authorisations state what a principal is permitted to do by using the actions defined by the authorisation. Authorisations can be shared between principals through roles or on the basis of direct assignments.

Obligation policies are an abstraction for defining the actions that must be performed by a principal on some target object when some specified event occurs. We extended the object model of [Damianou et al., 2001] and distinguish between general obligations that may be assigned to a role or a principle (e.g. a general obligation to process invoices from supply companies) and their specific instances (e.g. to process the invoice i1 from supplier x1). To support this distinction we require the following four rules to hold (these and are formally defined in [Schaad, 2004]):

1. An obligation instance must always relate to exactly one Principal.
2. An obligation instance has always one general obligation.
3. Every specific obligation a principal holds must be an instance of a general obligation he is a subject of through one of his roles or directly.
4. A general obligation can only have a principal or one of his roles as a subject, but not both.

The `s_subject` relation captures the assignment of a policy object to an object (which can be a role, principal or other policy object). Using an approach called objectification of state [Jackson, 2001], we can model total order relationships of states, where an expression like `(s1.s_subject).p1` would result in all the policy objects principal `p1` is a subject of in state `s1`.

The first rule mentioned above demanding the direct assignment of an obligation instance to a principal would thus translate as follows. Here `&` is the set intersection operator and `Principal` the set of principals, while the expression `obl.(s.s_subject)` yields all the principals subject to an obligation instance `obl`:

```
fact {all s : State | all obl : ObligationInstance |
one (obl.(s.s_subject) & Principal)}
```

For maintaining a history in such state sequences, we consider the following signature which maintains a DelegationHistory.

```
sig DelegationHistory{
 delegating_principal : Principal,
 receiving_principal : Principal,
 based_on_role : option Role,
 delegated_policy : PolicyObject
}
```

We do not maintain the information about which principal delegated which policy object in the form of an explicit relation, but in an explicit signature with several binary relations. We can do this because we know that, for example, in the context of the delegation of a policy, exactly one principal delegates exactly one policy object to exactly one other principal in between two states. This cardinality is indicated by the absence of the `set` keyword. The delegating principal may have chosen to delegate on the basis of a direct assignment or over a role as indicated by using the option keyword. Specifically this latter point could not be resolved in a n-ary relation like `Principal -> Principal -> Role -> PolicyObject` as there is no kind of null value in Alloy that would allow us to express that no role but a direct assignment was used for the delegation. A `RevocationHistory` and an `AccessHistory` signature have been defined in a similar way in [2] together with a set of rules, that, e.g. define that for any transition between states there can only be one history entry and other integrity preserving constraints. In essence, we can use Alloy to model sequences of states and define and analyse object access, delegation and revocation activities over such sequences. The history signature is then updated over the lifetime of such a sequence, and maintains, for example, the changes in the `s_subject` relation when moving from one state to the next as possibly initiated by a delegation activity. This provides all the information needed for supporting revocation activities.

3 Revocation of Policy Objects

In general, revocation of an object is based on its previous delegation and thus requires the following pieces of information [Samarati and Vimercati, 2001]:

- The principals involved in previous delegation(s);
- The time of previous delegation(s);
- The object subject to previous delegation(s)

Our conceptual model provides this information through the defined history signatures and may thus support the various forms of revocation as described in the revocation framework of [Hagstrom et al., 2001].

In this framework different revocation schemes for delegated access rights are classified against the dimensions of resilience, propagation and dominance. Since resilience is based on negative permissions, we do not consider this here, as there is no corresponding concept for the policy objects in our model (unlike Ponder [Damianou et al., 2001] which does provide negative authorisation policies).

The remaining two within our model dimensions may be informally summarised as follows:

1. Propagation distinguishes whether the decision to revoke affects
   - only the principal directly subject to a revocation (local); or
   - also those principals the principal subject to the revocation may have further delegated the object to be revoked to (global).
2. Dominance addresses conflicts that may arise when a principal subject to a revocation has also been delegated the same object from other principals. If such other delegations are independent of the revoker then this is outside the scope of revocation. If, however, such other delegations have been performed by principals who, at some earlier stage, received the object to be revoked via a delegation path stemming from the revoker, then the revoking principal may
   - only revoke with respect to his delegation (weak);
   - revoke all such other delegations that stem from him (strong).

Table 1. Revocation schemes

No	Propagation	Dominance	Name
1	No	No	Weak local revocation
2	No	Yes	Strong local revocation
3	Yes	No	Weak global revocation
4	Yes	Yes	Strong global revocation

Based on these two dimensions, we established 4 different revocation schemes which, due to the absence of the resilience property, are a subset of those described by [Hagstrom et al., 2001]. These are summarised in table 1. We will now investigate how far these schemes can be expressed and integrated with respect to our control principle model and the specific types and characteristics of policy objects. The following two sections will thus discuss the revocation of delegated policy objects along the lines of the above revocation schemes.

3.1 Revoking Delegated Authorisations

Since authorisations are similar to the notion of permissions in [Hagstrom et al., 2001], we will describe the four revocation schemes in terms of a possible delegation scenario. We use function `delegate_auth(s1, s2, p1, p2, auth)` to state that a principal `p1` delegated an authorisation `auth` to a principal `p2` in state `s1`. Similarly, `auth` was delegated by `p1` to `p3` in state `s2`; by `p3` to `p2` in state `s3`; by `p2` to `p4` in state `s4`; by `p2` to `p5` in state `s5`; and finally by `p6` to `p4` in state `s6`. This is summarised in the graph displayed in figure 1 where the nodes stand for the principals, and the arcs are labeled with the respective delegation activity. We assume that in this specific above example the principals always retain the authorisation they delegate. However, it must be noted that in general a principal might decide to drop an authorisation at the time

he delegates, which increases the complexity of delegation and revocation schemes as shown in [Schaad, 2003].

A weak local revocation of an authorisation is the simplest case as it does not propagate or dominate any other delegations of the authorisation. For example, if principal p2 revokes auth from principal p5, then p5 will not hold auth anymore. If, however, p1 revokes auth from principal p2, then p2 will continue to hold auth due to the delegation of auth by p3 in state s3.

A strong local revocation will address this later scenario, and if p1 strongly revokes auth from principal p2 locally, then p2 will not continue to hold auth, however, p4 and p5 will. The strong revoke by p1 will only result in p2 losing auth completely, because p3 had been delegated auth by p1 and then delegated it to p1. All delegations of auth to p2 stem from p1. If, for example, p2 strongly revokes auth from p4, then p4 will still hold auth because p2 has no influence on the delegation of auth by p6.

The weak global revocation addresses the revocation of policies which have been delegated more than once through a cascading revocation. Thus, if p1 globally revokes auth from principal p2 then this will result in p5 losing auth, but p2 and p4 will still hold auth due to the delegation of auth by p3 and p6 in s3 and s6 respectively.

Letting p1 revoke auth from principal p2 strongly and globally, auth will then not be held by p2 and p5 anymore, but p4 will still hold it due to the individual delegation by p6.

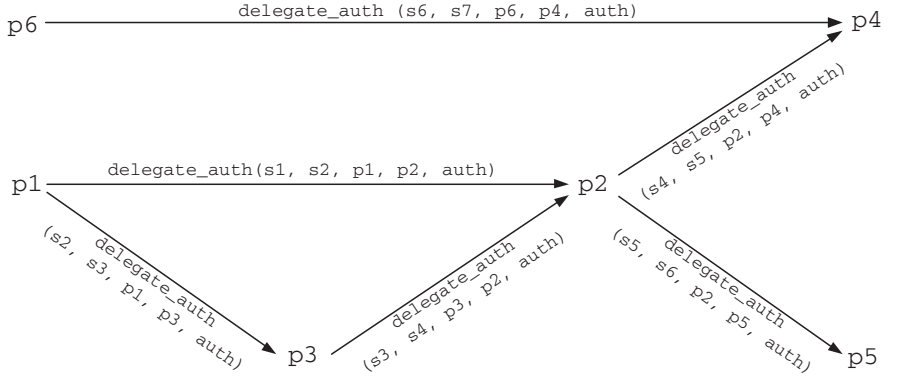

Fig. 1. A delegation scenario for delegating an authorisation auth

3.2 Revoking Delegated Obligations

The delegation of obligations should be complemented by revocation mechanisms as well, and we investigate in the following whether the previously identified four revocation schemes can also be applied in this context. Since we may delegate general and specific obligations as discussed in [Schaad, 2004], their revocation must also be discussed separately.

Revocation of Specific Obligations. We formally required an obligation instance to be assigned to exactly one principal at any time [Schaad, 2004]. It is for this reason that many of the problems we described for the revocation of authorisations cannot occur. We illustrate this with respect to the two applicable dimensions of revocation:

- Dominance does not apply as it is not possible for a principal to have been delegated the same obligation instance from different sources.
- Propagation may apply as a principal is able to delegate a delegated obligation. However, when he delegates he may not retain this obligation.

With respect to the second point, we consider the example of a principal `p1` having delegated an obligation instance `iob` to a principal `p2` who in turn delegated it to a principal `p3`. Should principal `p1` now be able to revoke `iob` directly from `p3` or only from `p2`? We believe that a principal should only be able to revoke a delegated obligation from the principal he delegated it to. The order of revocation thus corresponds to the way the obligation was initially delegated. There may be organisations where a direct revocation of `iob` from `p3` by `p1` may be desirable, e.g. a coercive organisation with distinct command structures, where decisions may have to be made rapidly like a hospital or military organisation [Mullins, 1999]. We do, however, not believe that any further argumentation would contribute to the overall goal of this paper.

Revocation of General Obligations. We have argued in [Schaad, 2004] that the delegation of general obligations can be treated almost identically to the delegation of authorisations. Thus, the underlying question here is whether this also applies to the revocation of general obligations. To clarify this, we again look at the revocation of general obligations with respect to the two dimensions of revocation considered in this context:

- Dominance applies since a general obligation may be held by several principals. These may independently delegate this obligation, perhaps to the same principal at different times.
- Propagation applies since a general obligation may be delegated several times between principals.

Considering the first item, the question is whether the revocation of a multiply delegated obligation may override other delegations or not. This has been referred to as strong and weak revocation respectively. We believe that this issue can be addressed like the strong and weak revocation of authorisations, as long as the defined constraints hold. The second item demands to distinguish between local and global revocation. The latter possibly causes a series of cascading revocations if a general obligation has been delegated several times. Again, this can be theoretically treated as in the case of revoking authorisations, but some additional points must be considered.

The delegation of a general obligation may have been followed by the delegation or creation of some instances of that general obligation. This may influence the revocation of a general obligation, because of the constraint that a principal may only hold an obligation instance if he has the corresponding general obligation. So if a general obligation is revoked, then this should result in the revocation of any existing delegated instances

for the principal subject to a revocation. A different situation may, however, be where an instance has been created on the basis of a delegated general obligation. The question is whether the revoking principal should be able to revoke an obligation instance he did never hold and subsequently never delegated. One approach may be to demand that the principal holding such instances must first discharge these before a delegated general obligation can be revoked. Another option may be to allow for the delegation of such an instance back to the revoking principal by the principal the general obligation is revoked from. Whatever the decision, we believe that these points present no technical difficulties with respect to the actual revocation activity. A more detailed discussion on organisational aspects of revocation is, however, outside this scope.

3.3 Defining Revocation Mechanisms for the CP Model

Several procedural revocation algorithms exist, as for example defined in the papers of [Griffiths and Wade, 1976], [Jonscher, 1998] or [Fagin, 1978]. However, the definition of a revocation mechanism in a declarative way is a non-trivial task and the only work we are aware of is [Bertino et al., 1997], which is unfortunately strictly tied to their specific authorisation model, and may also be difficult to understand without the possibility of tool supported analysis.

One main underlying design principle of our model is that a declarative specification of delegation and revocation operations should reflect their possible procedural counterparts. This means that they only cause a change to the `s_subject` relation with respect to the actual objects involved in the delegation and revocation. This has also been defined by a set of framing conditions that support each Alloy function. The strong local and the weak and strong global functions may however also cause changes to the `s_subject` relation with respect to other objects not explicitly defined when calling a revocation function, since this only describes what the resulting state should look like. We thus argue in the following, that the strong local, and weak and strong global revocation can all be modeled in terms of a series of weak local revocations. This weak local revocation supports revocation of authorisation and obligation policies.

For this reason we only formally outline the function `weak_local_revoke()` and point to the discussion in [Schaad, 2003]. The three remaining types of revocation are discussed less formally, as they may be understood as a series of weak local revocations.

Weak Local Revocation. Before considering the weak local revocation of a policy object in more detail, we informally recall some possible delegation scenarios that may have an effect on the behaviour of a weak revocation. These scenarios consider whether:

1. A policy object may have been delegated by a principal on the basis of a direct or role-based assignment; or whether
2. The principal a policy object was delegated to may have already been assigned with the object
   - either because he was assigned with the object at the time of system setup or;
   - because of a prior (and not yet revoked) delegation from some other principal.

Depending on the situation, a weak local revocation may behave differently with respect to the changes in the s_subject relation. For example, we consider that a principal p1 delegates an authorisation policy object auth he directly holds to a principal p2 in state s1. In this example, as a result of this delegation p1 loses auth. Principal p2 is also delegated auth by some other principal p3 in state s2. In state s3 principal p1 revokes auth from p2. Because of the delegation by p3 in state s2, p2 must not lose auth. Principal p1 must be assigned with auth again and the revocation of auth from p2 by p1 must be recorded by an update to the revocation history of state s3. A variation of this scenario may be that principal p1 delegated auth in state s1 on the basis of a role he is a member of. This would then mean that when he revokes auth from p2 in s3, the s_subject relation would not change at all. This is because of the delegation by p3 in state s2 and the fact that the initial delegation by p1 in s1 was based on a role, indirectly demanding the retainment of auth by p1 through his role membership. Nevertheless, the revocation would still have to be recorded by an update to the revocation history.

It would not be helpful to provide an exhaustive list of all such possible delegation scenarios here, and the two above examples only reflect in parts the complexity of a weak local revocation within our framework. There are, however, four general properties that need to be evaluated. These concern whether:

- there were multiple delegations of a policy object by and to the same principal;
- whether a role was used for delegation;
- whether there were multiple independent delegations;
- and whether the receiving principal was already subject to the delegated policy object before any delegations.

Based on these two dimensions and more general above properties, we established four different revocation schemes which, due to the absence of the resilience property, are a subset of those described in the revocation framework by [Hagstrom et al., 2001]. This subset is summarised in table 1 and the following four function headers 1-4 outline the behavior and expected return values.

These four functions are now composed to define the weak_local_revoke function 5. For reasons of space we only show the first half of this function. The revoking principal p1 must have delegated a policy object pol to p2 for any revocation to succeed (Precondition). If no role was used for the delegation (Case I) and the delegation was performed on a direct assignment instead, then we check whether there were no delegations of pol to p2 by other principals (Case I.1). This is sufficient as a principal must not delegate the same object twice without an intermediate revocation as defined in the precondition. If Case I.1 holds, then we check whether principal p2 held pol initially or not (Case I.1.a and I.1.b) and update the revocation history and s_subject relation accordingly.

If there were delegations by other principals (Case I.2) then we do not need to check for any initial assignments and just update the revocation history and s_subject relation. The second part of the function checks for the case of a role having been used for the delegation and is not shown here as it is similar in its structure to the first part. The full function and sequence of delegations and revocations we used to test and validate this weak local revocation with can be found in [Schaad, 2003].

Alloy Function 1. *The precondition of revocation. This function evaluates true if a revoking principal* `p1` *delegated the policy object* `pol` *to a principal* `p2` *in some state before the current state* `cstate` *and did not revoke* `pol` *from* `p2` *between that delegation and* `cstate`*.*

```
fun revocation_precondition
 (cstate: State, disj p1, p2: Principal, pol: PolicyObject) {...}
```

Alloy Function 2. *A role was used for the delegation. This function evaluates true if a principal* `p1` *delegated the policy object* `pol` *to* `p2` *in a state before the current state* `cstate` *on basis of a role-based assignment to* `pol`*.*

```
fun role_was_used_for_del_of_pol
 (cstate: State, disj p1, p2: Principal, pol: PolicyObject) {...}
```

Alloy Function 3. *The object to be revoked was delegated by some other principal. This function evaluates true if some principal* `p` *other than* `p1` *delegated the policy object* `pol` *to principal* `p2` *and did not revoke it before the current state* `cstate`*.*

```
fun pol_was_delegated_by_other_p
 (cstate: State, disj p1, p2: Principal, pol: PolicyObject) {...}
```

Alloy Function 4. *The principal* `p2` *a policy object* `pol` *is to be revoked from may have held* `pol` *even before any prior delegation by the revoking principal* `p1`*. This function evaluates true if principal* `p2` *held* `pol` *directly in the first state of a state sequence.*

```
fun rev_p_held_pol_initially
 (cstate: State, disj p1, p2: Principal, pol: PolicyObject) {...}
```

Strong Local Revocation. A strong local revocation would be almost identical in its specification. As in the function weak_local_revoke() we would have to check whether a principal was delegated the policy object to be revoked from some other principal. If this is true, like in the example in figure 1, and all delegations of an authorisation policy object `auth` to a principal `p2` stem from principal `p1` requesting the revocation, then `p2` will not be subject to `auth` anymore. On the other hand a strong local revoke of `auth` from `p4` by `p2` would have no effect on `p4`'s assignment to `auth` due to the previous delegation of `auth` to `p4` by `p6`.

These scenarios emphasize again the need for not only keeping track of delegated but revoked policy objects as well. Due to the underlying assumption of our model that only one revocation may happen at a time, such a strong local revocation function cannot be used directly, since it may change several relationships which we cannot keep track of. Nevertheless it may be used to assert that a certain sequence of weak local revocations would suffice for the definition of a strong local revocation. With respect to figure 1 this would mean that a sequence of weak local revocations of `auth` from `p2` by `p1` and `p3` should be equal to a single strong local revocation of `auth` from `p2` by `p1`.

Alloy Function 5. *Weak local revocation function composed of functions 1-4.*

```
fun weak_local_revoke (disj s1, s2: State,
                                        disj p1, p2: Principal,
                                        pol: PolicyObject){
 //Precondition: p1 has delegated pol to p2 before s1
   revocation_precondition(s1, p1, p2, pol) &&
 //Case I: No role was used for this initial delegation
   (!role_was_used_for_del_of_pol(s1, p1, p2, pol) =>
   //Case I.1: No other delegations occurred
     (!pol_was_delegated_by_other_p(s1, p1, p2, pol) =>
       //Case I.1.a: p2 did hold pol initially
         (rev_p_held_pol_initially(s1, p1, p2, pol)   =>
            update_rev_history(s1, p1, p2, pol) &&
            s2.s_subject = s1.s_subject + pol -> p1) &&
       //Case I.1.b: p2 did not hold pol initially
         (!rev_p_held_pol_initially(s1, p1, p2, pol)  =>
            update_rev_history(s1, p1, p2, pol) &&
            s2.s_subject = s1.s_subject + pol -> p1
                                        - pol -> p2)) &&
   //Case I.2: Some other delegation occurred
     (pol_was_delegated_by_other_p(s1, p1, p2, pol)  =>
         update_rev_history(s1, p1, p2, pol) &&
         s2.s_subject = s1.s_subject + pol -> p1)) &&
   ...
 //The following second part contains the same cases
 //if a role was used for the initial delegation.
   ...
}
```

Weak and Strong Global Revocation. Alloy has only recently started to support recursion, an indispensable mechanism to support global revocation as we described it in the previous section. At the time of writing our specification there was no available documentation or examples for the use of recursion. Nevertheless, we felt that at least an outline of how to define global revocation must be provided. The constraint analyser could provide us with a reasonable level of assurance about the working of a global revocation function as defined in [Schaad, 2003].

Weak and strong global revocations are similar in their effects to their local counterparts, however, they also consider any possible further delegations of the object to be revoked by the principal this object is revoked from. We have described this in the previous section and only want to point out some specific issues that need to be considered when defining such a global revocation.

Since recursion is required to provide for a global revocation, this means that the weak and strong global revocations functions consist of two parts. In the first part we check whether the object `pol` had been initially delegated by the revoking principal `p1` to `p2` as previously outlined. In the second part we then need to check whether

there was any further delegation of `pol` to some other principal `p`. If this is so, the global revocation function calls itself, now with `p2` being the revoking principal and `p` being the principal `pol` is revoked from. An example of such a recursive revocation is provided in [Schaad, 2003].

A series of weak local revocations may achieve the same result as weak and strong global revocations, but we did not investigate this any further considering formal proof, as there was not immediate need in the context of this paper.

4 Summary and Conclusion

In this paper we have provided a first possible approach to the revocation of policy objects in the context of our control principle model presented in [Schaad, 2003]. This approach followed the schemes as proposed in [Hagstrom et al., 2001], but due absence of negative authorisations and the specific notion of general and specific obligations and their respective assignment to roles and principals, not all categories of the scheme had to be considered.

We see our work as particularly useful in the context of workflow systems and their security, since our understanding of obligations as event-condition-action rules matches the notion of tasks. Our general obligations then refer to the tasks at the workflow model level, while specific obligations are the occurring instances at execution time of the workflow. The delegation of such tasks may then trigger the delegation of the required permissions.

However, there is remaining work. We need to further analyse the effect of revocation activities on existing review obligations. In particular, we would like to support revocation of an obligation instance by any principal in a delegation chain. Secondly, we specified in our framework that an obligation has a set of supporting authorisations. Although we did not fully investigate this relationship between authorisations and obligations in the context of delegation and revocation activities, we could observe that the delegation and revocation of authorisation objects may violate existing separation of duty properties [Schaad, 2003]. In particular, we could show how dynamic separation properties are "circumvented" by colluding principals with the right to delegate and revoke. This is not a new problem [Harrison et al., 1976] but still requires further analysis from a business process engineering perspective.

Overall, we have now completed the majority of our conceptual work and will look at the implementation of the concepts of delegation, review, evidence, revocation of general and specific obligations and the possible schemes and their practical feasibility in more detail. In fact, the work on collaborative workflows suggested in [Schulz and Orlowska, 2004] will offer interesting perspectives and SAP Research has already implemented collaborative workflow prototypes within which our organisational control principles and delegation and revocation schemes can be implemented. Together with analysis tools such as described in [Rits et al., 2005], we may then achieve a tight match between workflow tasks and the required permissions at application, middleware and database level.

References

Bertino et al., 1997. Bertino, E., Samarati, P., and Jajodia, S. (1997). An Extended Authorization Model for Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):85–101.

Damianou et al., 2001. Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The Ponder Policy Specification Language. In *Policies for Distributed Systems and Networks*, volume 1995, pages 18–38, Bristol. Springer Lecture Notes in Computer Science.

Fagin, 1978. Fagin, R. (1978). On an Authorization Mechansism. volume 3, pages 310–319.

Griffiths and Wade, 1976. Griffiths, P. and Wade, B. (1976). An Authorization Mechanism for a Relational Database System. *ACM Transactions on Database Systems*, 1(3):243–255.

Hagstrom et al., 2001. Hagstrom, A., Jajodia, S., Parisi-Presicce, F., and Wijesekera, D. (2001). Revocations - A Categorization. In *Computer Security Foundations Workshop*. IEEE Press.

Harrison et al., 1976. Harrison, M., Ruzzo, W., and Ullman, J. (1976). Protection in Operating Systems. *Communications of the ACM*, 19(8):461–471.

Jackson, 2001. Jackson, D. (2001). A Micromodularity Mechanism. In *8th Joint Software Engineering Conference*, Vienna, Austria.

Jonscher, 1998. Jonscher, D. (1998). *Access Control in Object-Oriented Federated Database Systems*. PhD thesis, University of Zurich.

Mullins, 1999. Mullins, L. (1999). *Management and Organisational Behaviour*. Prentice Hall, London, 5th edition.

Rits et al., 2005. Rits, M., De Boe, B., and Schaad, A. (2005). Xact: A bridge between resource management and access control in multi-layered applications. In *ACM Software Engineering Notes of Software Engineering for Secure Systems (ICSE05),*, St. Louis, Missouri, USA.

Samarati and Vimercati, 2001. Samarati, P. and Vimercati, S. (2001). Access Control: Polcies, Models and Mechanisms. In Focardi, R. and Gorrieri, R., editors, *Foundations of Security Analysis and Design*, pages 137–196. Springer Lecture Notes 2171.

Schaad, 2003. Schaad, A. (2003). *A Framework for Organisational Control Principles, PhD Thesis*. Phd, University of York.

Schaad, 2004. Schaad, A. (2004). Delegating organisational obligations - an extended analysis. In *IFIP WG 11.3 Database and Applications Security XVIII*, Sitges, Spain.

Schaad and Moffett, 2002. Schaad, A. and Moffett, J. (2002). Delegation of Obligations. In *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, Monterey, CA.

Schaad and Moffett, 2004. Schaad, A. and Moffett, J. (2004). Separation, review and supervision controls in the context of a credit application process, a case study of organisational control principles. In *ACM Symposium of Applied Computing*, Cyprus.

Schulz and Orlowska, 2004. Schulz, K. and Orlowska, M. (2004). Facilitating cross-organisational workflows with a workflow view approach. *Data & Knowledge Engineering*, 51(1):109–147.