9 Summary and Conclusions

Section 9.1 provides a short summary of this thesis. Section 9.2 evaluates the contributions made in this work and section 9.3 lists options for future research.

9.1 Summary

The main question addressed in this work is how the specification, deployment and management of application–oriented access control policies in distributed object systems can be supported in a way that increases the overall security. The first chapters of this thesis examine the problems that need to be addressed and identify a number of requirements for manageable access control. The overall management task is analyzed and structured into subtasks that are performed by potentially separate managers: principals or credentials management, object and domain management, and policy management. Also, the tasks of policy deployment and development are examined. As a result, we identified the requirements for documentation, support for communication between the involved parties, and for suitable management abstractions. It was concluded that an integrated approach to secure software development and management was required and that it could best be supported by the definition of a declarative policy language. Looking at the current technology for CORBA security revealed conceptual scalability problems and lack of structured support for policy design.

Therefore, this thesis proposes a new view–based access model and a declarative specification language called *view policy language* (VPL). The abstractions of this language are designed to support deployment and development as well as management of application policies. The central concepts of VPL are views as a first–class concept for the type–safe aggregation of access rights, roles as a task–oriented abstraction of callers, and schemas as a means of specifying triggered dynamic changes in the protection state.

To prove the practical relevance of these concepts, a comprehensive case study was analyzed and implemented. The presentation in chapter 6 showed examples of using roles, views, schemas, negative rights, and conditional views in the context of a conference paper reviewing system. The technical feasibility of view–based access control was shown through an implementation of the required security infrastructure, which included an interceptor–based access control mechanism, a language compiler, view and role repositories, and graphical management tools.

9.2 Evaluation

The approach to the design of a policy language that was taken here can be summarized as trading generality for type–safety and manageability. Fixing the data model for protected resources to CORBA IDL means that VPL is less general than other access models and that only remote CORBA invocations can be checked using this approach. Other policy specification languages and frameworks, e.g., Ponder [Damianou et al., 2001] and ASL [Jajodia et al., 1997], are generic and thus more widely applicable. However, approaches like these do not offer the same degree of type–safety: VPL definition constraints help to catch a large number of specification and administrative errors which cannot be detected without relying on the type system of the target object model. When compared with the standard CORBA access model, VPL offers a much greater flexibility and does not exhibit the same scalability problems.

The case study in chapter 6 revealed that even seemingly simple applications can have subtle protection requirements. VPL provides both expressive and flexible means to capture the semantics of access policies, especially through its schema construct and conditional views. These abstractions also help to limit the complexity of access control as a whole. VPL constraints, e.g., role restrictions, not only help to catch errors, they also support more descriptive policies that document the intention behind policies.

The main advantage of view–based access control over other approaches to protection examined in this thesis is its integrated approach to development, deployment and management of policies. VPL integrates with standard software development techniques such as UML models and can thus be expected to meet higher developer acceptance than authorization languages relying on specialized security models. Also, overall manageability is increased because VPL is not restricted to either policy design or management but provides a common language that enables communication within a single model between all involved parties. Such an integrative approach to access management is not provided by other high–level languages. Only EJB [Sun Microsystems, 2000] or CCM [OMG, 1999c] take a similar approach, but the descriptor languages of these technologies are less expressive and flexible than VPL and also do not provide the same degree of type–safety.

The discussion of the security infrastructure in chapter 7 showed that fine–grained protection induces a performance overhead that appears to be generally acceptable. The security infrastructure for VPL was designed to provide centralized management and decentralized enforcement of policies. This generally results in superior performance when compared to architectures relying on centralized decision servers, both because a centralized service is likely to become a bottleneck and because of the additional communication overhead for each decision. The prototype implementation has shown that this is a feasible approach. The advantage of using CORBA interceptors is that the enforcement mechanism is modular and configurable.

9.2.1 Application Areas

The answer to the question which kinds of applications actually benefit from view–based access control and which applications do not depends both on the granularity of the protected objects and on the nature of the policies that need to be enforced. An important factor in the design of realistic distributed applications is the cost of remote communication. Because remote invocations are approximately three orders of magnitude slower than local invocations it is a fundamental design goal to reduce the number of these remote invocations. Therefore, CORBA objects are typically not designed to be arbitrarily fine–grained and their interfaces tend to provide more operations than ordinary programming language objects.

As an extreme example, customized fine–grained data structures with simple operations that essentially only read and write are unlikely to be exposed as CORBA objects. Accesses to very fine–grained objects can be specified and later controlled using VPL, but the resulting view definitions are likely to contain only a single right per view, so some of the benefits of VPL like the task–based specification of roles, reuse through view extension and enhanced expressiveness when compared with generic rights are not as important as in other policy design situations. Type–safety is still an important advantage, however.

The benefits of using VPL are more apparent when view–based access control is applied to medium to large–grain CORBA objects that provide larger numbers of operations with richer semantics and are used in different use cases. The name server interface and the example policy in chapter 2 are representative of this granularity. Views are a useful abstraction for specifying access rights in a task–specific way because they can capture the richer semantics of accesses without the modeling problems of generic rights. Also, view extension can be used for reuse and modeling purposes.

VPL was specifically designed for application access policies. If the only resources that require protection in an application are files and database records that are already protected by the mechanisms of the underlying operating system or database management system, then the additional overhead of another protection mechanisms might not appear justified. However, if the application adds application–specific access rules of its own a separate VPL specification is still useful, even if the access policy can be enforced by existing, lower–level mechanisms. Without a separate policy document, however, the application's security requirements are specified only in a mechanism–dependent way and cannot easily be extracted and translated in case the application is ported to a different platform.

9.3 Future Work

While VPL as described and implemented here is currently restricted to CORBA, it could be applied to other distributed object platforms as well, e.g., to Java RMI [Sun Microsystems, 1997]. In this case, the object model would be that of the Java language and the type checks in the VPL compiler would have to be adapted accordingly. As pointed out in chapter 4, a problem that would arise for the static analysis of VPL texts is that the semantics of `strong` could not be statically guaranteed. Changes to VPL would also be necessary because of method overloading in Java, which is not allowed in IDL. A simple approach to dealing with overloaded names would be to adopt a name mangling scheme for access rights that would encode method parameter types in the name of the access right.

The view–based access model introduced in this thesis can be extended in a number of ways. An interesting and technically feasible option is the integration of content–specific access control rules into VPL. Content–specific access control enables policies that restrict accesses based on the value of an operation parameter or object attribute. As an example, consider policy rules such as "managers may give employees direct salary raises of up to 10%". VPL access rights definitions in views could be extended to include predicates on operation arguments and access decision objects would dynamically evaluate these predicates to check that an argument difference to an operation increaseSalary() does not exceed 10% of the current salary of an employee.

A second option for model extensions is to define operators for static policy combinations, in particular for refinement or specialization of policies and for composing policies from existing ones. Composing policies can be useful when applications are composed from building blocks, e.g., a new application wants to reuse an existing document management component and at the same time reuse its application policy design, i.e., its views and schemas. Rather than simply adding up the definitions of disjunct policies, composition would allow to identify roles and views from component policies. Refinement of policies could rest on importing and extending views and roles from existing policies and would result in specialized views and more powerful roles.

A promising field of research is the further integration of policy design with software development. In this thesis, use case diagrams were used as a starting point for the identification of roles and tasks. This approach could be extended to also use UML class and collaboration diagrams as a starting point for the identification or even automatic generation of views. [Koch and Brose, 2001]

A final area for future work is the application of VPL to access control mechanisms other than interceptors at the target. The most prominent candidate for such a mechanism are application–level firewalls, i.e., IIOP proxies such as Gatekeeper [Borland, 2000], Wonderwall [IONA, 2000], or Domain Boundary Controller [Xtradyne, 2001]. Given a number of VPL policies for protected objects in an interior network, an IIOP proxy could block requests at the network boundaries that would be rejected by interceptors anyway. Because accesses from within the interior network do not pass through a firewall, such an access control mechanism cannot replace access control at the target unless all accesses originating from the interior network are a priori trusted. However, firewall access control can complement access control at the target and potentially increase the overall performance by reducing network traffic and relieving target nodes from making access decisions. For example, target access interceptors could push negative results of access decisions to the firewall, which would simply reuse these decisions to filter out requests.