

# Integrating Security Policies via Container Portable Interceptors

Tom Ritter  
Fraunhofer FOKUS  
Kaiserin-Augusta-Allee 31  
10589 Berlin, Germany  
+49 30 3463 7278  
ritter@fokus.fraunhofer.de

Rudolf Schreiner, Ulrich Lang  
ObjectSecurity Ltd.  
St John's Innovation Centre, Cowley Road,  
Cambridge, CB40WS, United Kingdom,  
+44 1223 420252  
{rudolf.schreiner|ulrich.lang@objectsecurity.com}

## ABSTRACT

In the past, it was very common to develop middleware without consideration of security from the very beginning. To integrate security, the middleware that should be protected has to provide appropriate hooks and interfaces, and has to meet the requirements of the security architecture. In most cases it is not practical to develop a new, secure middleware from scratch. It is only possible to make minor modification to existing systems. In this paper we describe the successful integration of a CORBA Component based middleware and a policy management framework for the definition, management and enforcement of security policies. Integration is achieved by defining Container Portable Interceptors and QoS Enablers which provide the necessary hooks for interception and provision of context interfaces to integrate the security framework. To practically evaluate our concepts, we also describe how our reference implementation is used for the development of an experimental secure Air Traffic Control system.

## Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Client/Server, Distributed applications

D.4.6 [Security]: Access controls, authentication, information flow controls

## General Terms

Management, Security

## Keywords

CORBA Portable Interceptors, Policy Management Framework, CORBA Component Model, Container Portable Interceptors

## 1. INTRODUCTION

Enforcing appropriate security policies in distributed, component based applications is a difficult task. In many cases, the security functionality of standard middleware like CORBA or Enterprise Java Beans is not sufficient to protect an application and its resources. Today the common solution for this problem is still to directly implement security enforcement in the application source code. While this approach is very flexible, there are several serious issues. First of all, it requires that the persons designing and implementing the application components are security specialists, too. In real life, this is rarely the case. Sometimes the

component developers also do not know in which environment the component will be used and are therefore unable to define an appropriate security policy. So it is common that the application programmers simply ignore security. The second issue is the coupling of functional and non-functional aspects, which hinders component reusability. A component can only be reused if the functional and non-functional requirements match.

Therefore it is necessary to clearly separate functional and non-functional parts of an application. The functional parts, i.e. the business code, are implemented in a component. The non-functional parts, for example the enforcing of the security policy, must not be mixed with the business code. These have to be defined by a separate policy and to be enforced by the runtime infrastructure of the component.

While this sounds good in theory, it is hard to do. First of all, a generic framework to define and evaluate security policies is needed. Secondly, it is necessary to integrate the security framework with the middleware platform. The middleware has to provide the necessary hooks to intercept calls and to obtain the information required for security enforcement.

In the following, we describe the middleware platform we use, the CORBA Component Model (CCM). We then describe our security framework and its integration into CCM in detail and finally evaluate it in a real world project.

### 1.1 CORBA Components

The CORBA Component Model (CCM) [1] is one of the best platforms for developing large scale distributed systems. It is based on the mature CORBA middleware [2] and includes several more advanced concepts. It also simplifies the usage of the CORBA Services.

CCM enhances the Object Model of CORBA. Figure 1 depicts the features a CORBA Component can have. A component has a special interface (equivalent interface). This interface provides operations for introspection and navigation regarding other component features. A component can provide a set of facets. A facet is a named port providing a specific interface. Clients of this component call operations on a facet. The facet's counterpart, a receptacle, is a named port where a specific interface can be connected to. A facet of a CORBA Component in server role can be connected to a receptacle of a CORBA Component in a client role. Receptacle ports make dependencies to other interfaces explicit, which helps to minimize wrong configurations and run-

time failures by providing type safety. As facets and receptacles are used for operational interactions, event sources and event sinks are used for asynchronous interaction. An event source can publish or emit events of a certain type. Event sinks can consume events of a certain type. Finally, a similar port concept for continuous interactions (i.e. data streams) has been introduced by the OMG to the CORBA Component Model as well. A stream source port produces streams of data of a specific type while a stream sink port can receive such data. Attributes can be used to configure an instance of a CORBA Component.

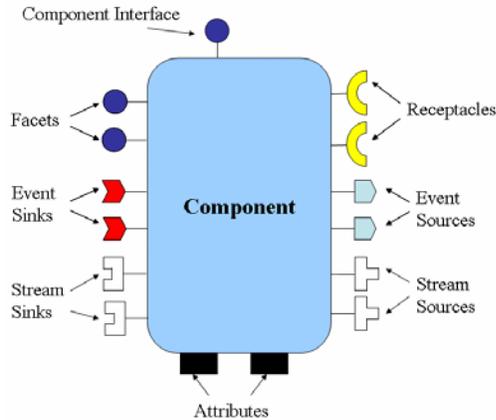


Figure 1. Object Model of CCM

CCM defines the container as the run-time environment of a CORBA Component (see figure 2), including an interface to the component which is called context, that provides access to the underlying platform services (e.g. CORBA Services) as well as access to the application environment of the component (e.g. connected interfaces at receptacle ports). Furthermore, the component implementation provides a call-back interface to the container which is in turn used to manage the life-cycle of the component implementation. In the container also resides a Home, which is an implementation of factory and finder pattern for managing component instances.

```
{ SHAPE \* MERGEFORMAT }
```

Figure 2. Container of CORBA Component

## 1.2 OpenPMF Policy Management Framework Overview

In previous work we established that the CORBA Security Services specification does not meet the requirements of complex, distributed systems [3, 17]. To overcome the limitations of CORBA security for CCM based applications, we developed the OpenPMF policy management framework [4] to define, manage and enforce security policies in complex distributed systems. While this paper is focused on our CORBA/CCM middleware platform, OpenPMF has been designed to also protect other platforms and applications.

OpenPMF is based on an abstract model of middleware security policies, defined in UML. From the abstract model we generated a policy repository to store concrete instantiations of security policies. Rich policies are defined in a consolidated way using a policy description language (PDL) (see [5] for PDL example excerpt). The policies are fed into the repository using a PDL compiler. During the application startup or policy updates, the security agents, also called Policy Enforcement Points (PEP), in the application obtain the policy from the repository and instantiate it. During runtime, the security agents intercept the invocations and evaluate the policy, based on the invocation's context. If the invocation is not allowed by the policy, the invocation is aborted and an exception is raised. In the following we will describe in detail how context information is described and obtained, and how policies are defined and enforced.

Due to space restrictions, we will only very briefly cover issues related to assurance, e.g. how we ensure that the policy is correctly enforced or how the integrity of the security framework itself is protected. We also will not discuss the principal limitations of security enforcement at the middleware level. Our objective is access control on invocations at the middleware level. This is a very important aspect in secure distributed systems, but in the development and operation of such systems many other aspects have to be considered as well, for example it has to be ensured that the middleware level protection cannot be circumvented by attacks at another level.

## 2. Context Information

A very important point in adaptive and reflective systems is the description of context. The context gives information about the invocation or the environment, and is used by the policy evaluator to make its decisions. It consists of information describing both the functional and non-functional aspects of the system. Functional information for example includes the target of the call and the operation to be invoked, which are both directly associated with an invocation. Non-functional information is in many cases also directly associated with an invocation, for example the client's credentials; in other cases it describes the environment independently of individual invocations, for example time or a position.

One of the biggest issues in the description and processing of context information is the lack of standardization of its format and semantics, mainly because it describes very different information from different sources. It is therefore very hard to define a standard and orthogonal format for the context. If the format is well defined and restrictive it can be handled in a standard manner, but it can only express very limited information. If on the other side the format is very flexible, for example a buffer, it can express very different information, but it is difficult to process in a uniform way. Another point to consider here is not only the data itself, but also the description of the type.

Context information is obtained from different sources, for example the client's credentials are provided by the underlying security mechanism, while the invocation target and operation have to be provided by the middleware platform. This complicates obtaining the context information a lot.

The different sources and formats of context information make its processing in policy management frameworks difficult, because it is impossible to foresee all possibilities in the policy evaluator. In

OpenPMF we developed an orthogonal approach for the handling of context information called Transformers.

The Transformers provide a uniform interface to the underlying data sources. To obtain data from different sources, the policy evaluator now can call the same function, whether the data is provided by the middleware, the security mechanisms or by any other source.

A standard interface to context information does not solve the problem of the information format and semantics. The policy evaluator still does not know how to process this information, for example how to compare the data. We solved this issue by moving the processing to the Transformer itself, because the Transformer knows the data it processes. Instead of letting the policy evaluator obtain the information from the Transformer and compare it by itself with a selector from the policy, the policy evaluator just calls the Transformer's compare function with the selector as an argument. The context information is obtained automatically by the transformers during the comparison operation.

The Transformer abstracts from all peculiarities of the underlying platforms, the different data sources, data formats and semantics. This greatly simplifies the policy evaluation and increases the flexibility of OpenPMF.

### 3. Policy definition and evaluation

In most security systems, security policies are defined as access control lists or using policy definition languages like Ponder. In these systems, the language is a very central aspect of the system.

In our policy management framework, we use a different approach. The central aspect is not the language, but the abstract information model of the policy, the *meta-policy*. This meta-policy gives an abstract way to describe policies, completely independent from how the policy is expressed (for example in a file). For the definition of the meta-policy we used the OMG's MetaObject Facilities (MOF) [6], and defined the meta-policy as an UML model. Our meta-model is very flexible and describes in a consolidated, unified way how to express policy hierarchies, rules and the entities used for the rule definitions, for example initiators, clients, targets and operations. This allows also the handling of security policies based on different security models like discretionary access control (DAC) [7], mandatory access control (MAC) [8] or role-based access control (RBAC) [9] by mapping the high level security policies to low level rules of the meta-policy.

From the meta-policy, a CORBA based policy repository is generated. The mapping of the model to the IDL interfaces is defined by the OMG MOF standard. The meta-policy and the repository derived from it do not solve the problem how to define a policy. For this purpose we developed a simple policy description language (PDL). It is able to describe policy hierarchies and policy rules. In contrast to common trends, we do not use XML for the policy description, because it was the initial intention to manually write the policy. In the next OpenPMF version, PDL will only be used as an internal representation for the GUI, and XML will be supported. The policy, expressed in a PDL text file, is compiled and loaded into the policy repository. Vice versa, it is also possible to generate a PDL file from the policy stored in the repository, or to visualize its logical structure.

The protection of the policy repository itself is a key issue of the assurance of the overall security framework. Using OpenPMF for protecting the repository is impossible; it would just generate a recursion. So we hardcoded the access control for the repository in the repository code itself and bootstrap the secure loading of policies via command line arguments.

The policy enforcement is done by *Policy Evaluators*. They are integrated into the middleware's call chain, for example in CORBA they are called by Portable Interceptors and in components by Container Portable Interceptors (COPI). At application startup, the Policy Evaluator registers itself with a central Management Daemon, obtains the security policy from the Policy Repository and instantiates it in an efficient and compact internal representation. After the initialization, the Policy Evaluator is automatically called for all invocations. It iterates through the internal policy representation and calls the Transformers associated with the entities in the rules, for example to check the client's identity. When a matching rule is found, the appropriate action is triggered, in most cases this means that the invocation is allowed. If no match is found, the invocation is denied by default, an exception is raised, and a predefined operation can be called. Policy Evaluator can be used both at the client and server side of an invocation, depending on the security policy to enforce.

A very important aspect of secure distributed systems is runtime monitoring. In OpenPMF, a central monitoring system is used. All events are sent to a central administrator console, where appropriate action can be taken. The management console also provides a runtime management of the security policies and their enforcement, e.g. it is possible to update security at runtime.

Meta-policies, model based and adaptive policy definition, and enforcement are powerful techniques for the development of secure and complex distributed systems. But this is very hard to implement in practice if the underlying middleware does not support sufficient interfaces for obtaining context information or the modification of the calls. As described above, this issue already made access control at the middleware level in CORBA very difficult. It is therefore necessary to analyze already during the development of the middleware how to provide the necessary hooks. In the following we will describe the Container Portable Interceptors (COPI). They are the result of a close cooperation between the container developers and the developers of the adaptive security architecture, which requested many additional features.

### 4. Container Portable Interceptors

As outlined before, it is crucial for evaluating and enforcing security policies to have access to the context, and in particular to have access to the call chain. It is important to have the hooks for putting a security policy in place. The fundamental requirement be able to intercept the call chain (which is not unique to security policy integration) led to the definition of Portable Interceptors (PI) as part of the CORBA specification [2]. PIs work on unmarshaled messages. This implies that they can't be used for messages encryption. Secure transport connections like SSL can be used instead.

The CORBA specification defines interfaces for the interceptors and for registering them with the ORB. There are a number of interception points defined which are called at specific points in

the call chain. Those interception points are also divided into client-side and server-side interception points. Figure 3 gives an overview of the Portable Interceptors.

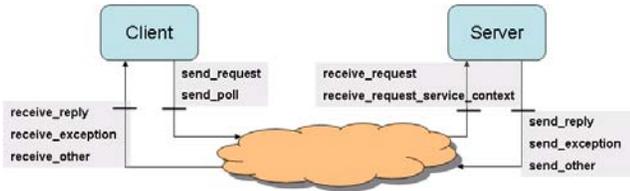


Figure 3. Interception Points

With the definition of the CORBA Component Model the CORBA Portable Interceptors can not be easily used for hooking user dependent code into the application logic. This is because the PI specification requires the registration of interceptors as part of the initialization process of the ORB. This process, as well as the complete management of the CORBA call chain, is done by the container. Even worse, the ORB creation is done before any user dependent code gets loaded into the container.

Two strategies are available to overcome this issue: Either add proprietary extensions to the container, or adapt the Portable Interceptors to the CORBA Component Model. The authors have decided to follow the second strategy. Initiated by the COACH project [10] and with the help of the participating partners a draft definition of the Component Portable Interceptors (COPI) was done. The result is currently in a standardization process and is supported by other companies as well [11].

There are two main goals to be achieved with the definition of COPI. The first one is to allow the smooth transformation of CORBA applications based on Portable Interceptors to CORBA Component based application. This means that, similar to the migration from CORBA to CCM, the easy migration from PI to COPI should be possible. The second goal is to extend the concepts of Portable Interceptors to provide more flexibility than plain CORBA Portable Interceptors give. CORBA Portable Interceptors have one fundamental design feature which renders them unusable for a very flexible and adaptive system architecture: they are not able to modify the call chain. This means they can only monitor the call, abandon a call or use Forward Location exception to modify the target of a call. Portable Interceptors can not be used to modify operation parameters.

To achieve both goals while not putting to many constraints on container vendors the Container Portable Interceptors are separated into a basic and an extended part. The basic part covers the same functionality as the plain CORBA Portable Interceptors do and the extended part offers additional functionality for modifying requests.

Since basic COPIs are meant for migrating PI code to component level, COPI interception points should be called at the exact same location as PI interception points would be called. This is easily achievable since CCM is designed to be on top of plain CORBA products. This means that for implementing the basic COPIs it is a natural decision to use the PI as foundation. Furthermore, COPIs have method signatures very similar to the ones of the PIs.

In contrast, the extended COPIs offer similar but different interception points. While basic interception points are called

while the ORB is dispatching a call, the extended interception points are called while the CCM container is dispatching a call. This is schematically depicted in figure 4.

However, there is an important difference in the flow-rules between basic and extended interceptors. The extended interceptors can for example return a result for a method call and can prevent the further call processing from reaching the component implementation. This gives the opportunity to modify the behavior of a component by providing different behavior without changing component implementation.

The definition of the COPI interface does not prescribe in any way how to implement the COPI interface and how to register the interfaces within the container. The specification [11] offers a way to do that by using an extended container category and the QoS Enabler concept. This is an optional way of doing it. Vendors can use other approaches to support the usage of COPI interfaces in their containers. The QoS Enabler concept, which is also used for implementing the hooks for integration of security policies, is explained in the next section.

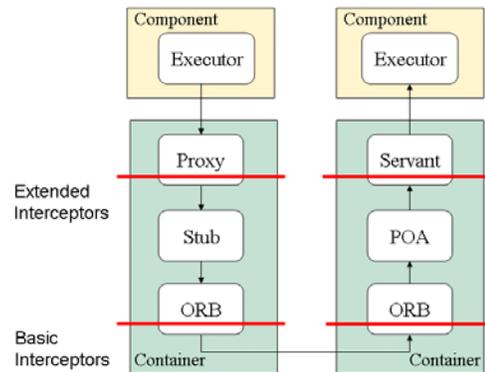


Figure 4. Basic and Extended Interceptors

## 5. QoS Enabler

The CORBA Component Model has defined the container model for providing a high level of abstraction to the component implementation. It also offers the possibility to load and to unload user code (components) dynamically by installing or de-installing Homes. Depending on mechanism used by the container vendor and programming language this is realized by loading and unloading shared libraries and it requires sophisticated management of such artifacts at run-time. This feature is implemented in Qedo [12].

The same principle is applied to the implementation of the COPI interface by defining the QoS Enabler concept. A QoS Enabler is a specialized component that can be loaded into a specialized CCM container (extension container) and is able to hook in additional functionality. Taking this approach allows the use of plain CCM mechanisms for development and deployment of QoS Enablers. The extension container offers the possibility to register COPIs with the container. Each QoS Enabler is responsible for a specific QoS category. In our case, we used the QoS Enabler concept to implement access control. Here, the QoS Enabler implements the Policy Enforcement Point. During initialization, it loads and instantiates the security policy. At runtime, it intercepts

the invocations, calls the Policy Evaluator and, if the invocation is not permitted, raises an exception.

The only condition that has to be ensured at run-time is that on each relevant node a corresponding QoS Enabler is instantiated. The QoS Enabler itself will furthermore be in contact with the policy framework to get an update on the security policies that have to be enforced and to centrally log security events like policy violations.

At run-time, the QoS Enabler at the client side checks the identity of the calling component. This is done at the interception point *send\_request*. It adds a security context to the call context. At the server side the QoS Enabler is called at interception point *receive\_request*. The QoS Enabler checks the origin identity and the target identity and checks whether there is a policy that has to be applied for that call.

QoS Enablers are components, and can be loaded, removed and configured like normal application components. This of course raises security issues, e.g. if the QoS Enabler itself is unprotected, an attacker could simply disable the policy enforcement by removing the OpenPMF QoS Enabler. In our implementation, this is not possible, since the complete underlying infrastructure is protected as well by the OpenPMF policy management framework.

## 6. Building a Secure System: Experimental Air Traffic Control Application

So far, we described how to integrate the definition, management and enforcement of security policies into the CORBA Components Model. In the following we will evaluate our concepts in a concrete project.

The goal of this project, called AD4 [13], is the development of an innovative 3D visualization system for Air Traffic Control (ATC). The visualization system itself is implemented using CCM and is integrated with existing ATC simulation systems, Eurocontrol Escape and Vitrociset ATRES. The functionality of these simulation systems and other data sources like weather data is also wrapped into CORBA Components.

From an architectural point of view, AD4 is very similar to future Network Centric Air Traffic Control and Collaborative Decision Making systems, and also shares many similarities with military Network Centric Warfare. It integrates data from different sources and organizations, and visualizes them for a human decision maker. This raises several security concerns, because in such future large and highly integrated systems of many organizations with different security standards and from various countries, there is a high risk of attack. Therefore it was decided to develop the AD4 system as a prototypical secure application. While this was initially planned as an academic exercise, during the course of the project it turned into reality, because the different simulation platforms are running at different project partners and have to be used securely over the Internet, reflecting quite exactly the real architecture and environment of the target applications.

As middleware platform for AD4, we use SecureMiddleware [14], our reference implementation of the concepts described here. It consists of Qedo, an enhanced implementation of the CORBA Components Model with QoS support, MICO [15], a CORBA ORB with improved support for security, and the OpenPMF Policy Management Framework with underlying security

infrastructure like a PKI and directory services. In addition to the CCM runtime platform, SecureMiddleware also includes a model driven development tool chain based on the OMG's Model Driven Architecture. This tool chain greatly reduces the complexity in designing the system and in defining security policies by raising the level of abstraction. It is possible to define application components and access control policies in a platform independent way by using UML2. A model transformation builds platform specific models (CCM models and security models) where additional refinement of these models can be done later by using a graphical modeling tool. Additional model transformations create programming language specific artifacts and specific security rules.

Before the security architecture for AD4 can be developed, it is necessary to define the security requirements. While in many secure systems, esp. in the military domain, confidentiality plays the most important role, in our system it is a lesser concern. The most important security objectives are availability and integrity. First of all, internal or external attackers have to be prevented from disrupting the correct operation of the system, e.g. by crashing it. It has also to be ensured that no attacker is able to spoof tracks or to modify other data.

The integrity of information is protected by means of access control. Only authorized sources are allowed to feed data into the system, and we also control the access of human users to the system. This is done using OpenPMF access control policies. For operative interfaces we use the RBAC model. For event based communication, which implements the main information flow in the system, MAC would be a possible choice. But since our system does not process confidential information we just allow certain flows using access control rules without labels or clearances.

A very important aspect here is the correctness of the security policy. If an interaction is not explicitly allowed by the policy, it is automatically denied by the Policy Evaluator. In an ATC system, this can lead to very serious consequences. As we learnt during the development of the system, the manual definition of the security policies is a very difficult and error prone process. Human security administrators are not able to define the security policy with a sufficient level of assurance, because of the high overall complexity of the system and its interactions. For this reason, we integrated the policy definition into the various steps of the model driven engineering process and automatically generate the security policies from the functional model of the application, using a model transformation.

The QoS Enablers and OpenPMF provide access control on the interfaces of the components and their infrastructure. This means for example that an attacker cannot successfully inject false tracks into the system or reconfigure it. But this security enforcement at the endpoints does not protect from Denial of Service (DoS) attacks; an attacker would still be able to overload the system. Full protection against such attacks is impossible. To reduce this risk, we use the concept of segmentation of the system into different domains. Within a domain, which is for example a control centre, the risk of a DoS attack is low and could be easily countered, since the physical access to the local systems is limited. Between the domains, here also between the organizations and the Internet, a Domain Boundary Controller (DBC) is used. This DBC, the ObjectWall IIOP Proxy [16], is

also integrated with OpenPMF and forwards all legitimate requests which are authenticated and authorized according to the security policy at the DBC. An attacker is now still able to launch a DoS attack against the DBC and stop the communication between the domains, but cannot attack the core functionality inside a domain anymore.

After some smaller tests, a first version of the AD4 application with simplified business logic was implemented and installed at the project partners' sites, in order to practically evaluate the communication, middleware and security infrastructure.

A very important aspect in distributed systems is performance. A security system with a too big impact on the overall performance is not really usable. In the current version of SecureMiddleware, we considered performance optimizations at the architectural level, but not really at code level. We also did not yet run extended benchmarks, which is a very difficult task anyway. Some very preliminary tests, mainly empty calls with a default crypto suite, showed that encryption, esp. the authentication, does have a considerable influence on the overall performance of the system, esp. on the CPU time, which is doubled. The additional effect of the Portable Interceptors and the policy evaluation is low. The CPU time used is increased by several percents, while the difference in the wall time is about 10-15%. In our current test applications the security enforcement did not raise any performance issues so far; therefore we did not yet implement optimizations of the code.

Apart from the abovementioned and now resolved issue related to correct policy definition, the policy management and enforcement at the QoS Enablers worked as expected and met the initial requirements. It was possible to define and enforce appropriate policies of sufficient granularity, to centrally manage the policies and Policy Evaluators in a large scale distributed system, and to monitor all relevant events.

## 7. Related Work

Many concepts described in this paper are based on previous and related work in middleware security, both in ongoing research and in already existing standards and implementations. For example the usage of Portable Interceptors for implementing access control is widely used in CORBA security [3]. [17] gives an overview over many different approaches in middleware security with a focus on access control and CORBA.

## 8. Conclusion

The separation of functional and non-functional aspects is crucial when developing reusable components and secure systems, but is very hard to fully achieve with today's platforms. We presented an integrated secure middleware based on extended CORBA Component Model and the OpenPMF policy management framework. QoS Enablers are used to provide implementations of the Container Portable Interceptors and the hooks required for the security enforcement. Policies can now be defined and updated independently of the component, at application startup and at runtime.

Our secure middleware was evaluated in several test applications and is currently successfully used in an experimental air traffic control system, which is deployed across the internet. It meets all initial design requirements; we were able to define, manage and enforce appropriate security policies for the components and infrastructure. Together with other security infrastructure like the

ObjectWall IOP Proxy, it is an important building block for the development of secure systems. However, the manual definition of security policies turned out to be cumbersome. This issue was improved by integrating security into the model based software engineering process, resulting in automated and assisted policy generation. The current snapshot of our system is mainly targeted at relatively static applications, as it does not yet fully meet the requirements of highly dynamic systems. In the nearer future, we plan to add policy based adaptation.

## 9. References

- [1] OMG, "CORBA Component Model", OMG document number formal/02-06-65
- [2] OMG, "Common Object Request Broker Architecture" OMG document number formal/04-03-12
- [3] Lang, U., Schreiner, R. Developing Secure Distributed Systems with CORBA, ISBN 1-58053-295-0, Artechhouse, 2002
- [4] ObjectSecurity, OpenPMF project, { [HYPERLINK "http://www.openpmf.org"](http://www.openpmf.org) }
- [5] Lang, U., Schreiner, R., Integrated IT Security: Air-Traffic Management Case Study, Proceedings of the ISSE Conference, September 2005
- [6] OMG, "MOF 2.0" final adopted specification, OMG document number ptc/03-10-04
- [7] Bell, D., LaPadula, L., Secure Computer Systems: Mathematical Foundations and Model. MITRE Report MTR, 2547, v2, 1973.
- [8] Sandhu, R.S. and Samarati, P. Access control: Principles and practice. IEEE Communications, Vol. 32, No. 9, pages 40-48, 1994.
- [9] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C. Role-Based Access Control Models. IEEE Computer, Vol. 29, No. 8, pages 38 - 47, 1996.
- [10] COACH Consortium. Component Based Open Source Architecture for Distributed Telecom Applications. (<http://www.ist-coach.org>). May 2003
- [11] OMG, "QoS 4 CCM" final adopted specification, OMG document number ptc/2006-04-15
- [12] Qedo - QoS Enabled Distributed Objects, <http://www.qedo.org>
- [13] AD4 Consortium. EU FP6 R&D project AD4 - 4D Virtual Airspace Management System, { [HYPERLINK "http://www.ad4-project.com/"](http://www.ad4-project.com/) }
- [14] ObjectSecurity & Fraunhofer FOKUS, SecureMiddleware project, [www.securemiddleware.org](http://www.securemiddleware.org)
- [15] MICO CORBA project, [www.mico.org](http://www.mico.org)
- [16] ObjectSecurity, ObjectWall IOP firewall, [www.objectwall.com](http://www.objectwall.com)
- [17] Lang, U. Access Policies for Middleware Ph.D. thesis. University of Cambridge. Cambridge, UK, 2003

## **10. CVs**

### **10.1 Tom Ritter**

Tom Ritter graduated with a Masters degree in Computer Science from the Technical University of Berlin. Since 1998 he worked at Fraunhofer Institute FOKUS in the area of tool development and distributed systems. His major interest is the development of component-oriented middleware platforms with consideration of extra-functional properties and the development of model based tools. In his recent work he developed a CORBA Component based Middleware Platform (Qedo). Tom is involved in various standardization activities at the Object Management Group and has contributed to workshops and conferences.

### **10.2 Rudolf Schreiner**

Rudolf Schreiner received his Master's Degree (Dipl-Phys.) in physics from University of Munich in 1993. In 2000, he became one of the two founders and chief technology officer of ObjectSecurity Ltd. Rudolf's main interest is the development of secure distributed systems in critical domains. He is the principal developer of OpenPMF and ObjectWall.

### **10.3 Ulrich Lang**

Ulrich Lang received his Ph.D. in information security from the Security Group of the University of Cambridge Computer Laboratory in 2003, and his Master's Degree (M.Sc.) in information security from Royal Holloway College (University of London) in 1997, after having studied computer science with management at the University of Munich and at Royal Holloway College (University of London). In 2000, he became one of the two founders and chief executive officer of ObjectSecurity Ltd. His current main interest is the reduction of the complexity of security administration in distributed systems through improved usability and visualisation.