Role-based Security for Configurable Distributed Control Systems

Michael Hauf, Janek Schwarz, and Andreas Polze
Department of Computer Science
Humboldt University of Berlin
Rudower Chaussee 25, 12489 Berlin, Germany
{hauf,schwarz,apolze}@informatik.hu-berlin.de

Abstract

The Common Object Request Broker Architecture (CORBA) is the most successful representative of an object-based distributed computing architecture. Although CORBA simplifies the implementation of complex, distributed systems significantly, the support of techniques for reliable, fault-tolerant, and secure software is very limited in the state-of-the-art CORBA. Extensions, such as the CORBAsec specification provide only a limited choice of coarse-grained mechanisms to specify access rights for components.

Any fault-tolerance or security extension for CORBA components needs to trade off data abstraction and encapsulation against implementation specific knowledge about a component's internal behavior, resource usage, interaction and access patterns. These non-functional aspects of a component are crucial for the predictable behavior of security and fault-tolerance mechanisms. However, in contrast to CORBA's interface definition language (IDL), which describes a component's functional interface, there is no general means to describe a component's non-functional properties, such as security settings or fault-tolerance.

Within this paper we present our approach towards role-based security for CORBA. Following the idea of aspect-oriented programming, we have developed a description language for security settings. The description language uses the eXtended Markup Language (XML) as an underlying representation and allows specification of access rights independently from an object's implementation. A role-editor tool allows for configuration of a component's security settings without affecting the component's source code. Security settings are enforced by our enhanced version of the ORBacus CORBA implementation.

We discuss the necessary description and configuration steps for a secure CORBA service. We demonstrate how our previously developed distributed tele-laboratory application can be configured for secure access. One may notice that, although this configuration step required the development of a number of role descriptions, no modifications to the tele-laboratory's source code were necessary.

Key words: CORBA, security, component configuration, aspect-description, XML.

1. Introduction

Computer systems used in today's airplanes, cars and automated factories (process control) are often implemented as special purpose systems which are vendor-specific, expensive, hard to maintain and difficult to upgrade. Often, those systems apply proprietary techniques to achieve security and predictable timing behavior, even in case of faults. With the need of integrating multiple of those control systems into a bigger whole, requirements arise to open up proprietary systems for standard (non real-time) distributed computing technology.

The Common Object Request Broker Architecture (CORBA) [6][7] is the most successful representative of an object-based distributed computing architecture. Although CORBA simplifies the implementation of complex, distributed systems significantly, the support of techniques for reliable, fault-tolerant, and secure software, such as group communication protocols or replication is very limited in the state-of-the-art CORBA or even fault-tolerant CORBA. Extensions, such as the CORBAsec specification provide only a limited choice of coarse-grained mechanisms to specify access rights for components.

Any fault tolerance or security extension for CORBA components needs to trade off data abstraction and encapsulation against implementation specific knowledge about a component's internal timing behavior, resource usage, interaction and access patterns. These non-functional aspects of a component are crucial for the predictable behavior of real-time and fault-tolerance mechanisms. However, in contrast to CORBA's interface definition language (IDL), which describes a component's functional interface, there is no general means to describe a component's non-functional properties, such as security settings, fault-tolerance measures and timing behavior.

Within this paper we present our approach towards role-based security for CORBA. Following the idea of aspect-oriented programming [5] we have developed tools and a description language for security settings. The description language uses the eXtended Markup Language (XML) as an underlying representation and allows specification of access rights independently from an object's implementation. In contrast to the CORBAsec specification developed by OMG, our security description language allows specification of access rights not only on the interface (class of objects) level, but on the single object instance level.

We discuss the necessary description and configuration steps at design-time, configuration-time and runtime for a secure CORBA service. We demonstrate how our previously developed distributed tele-laboratory application can be configured for secure access. One may notice that, although this configuration step required the development of a number of role descriptions, no modifications to the tele-laboratory's source code were necessary.

The remainder of the paper is organized as follows: Section 2 presents related work. Section 3 introduces our description language for the security aspect. The previously developed tele-laboratory application is described in Section 4, whereas Section 5 discusses the configuration steps necessary to extend the tele-laboratory with security features. Section 6 finally summarizes our conclusions.

2. Related Work

2.1 CORBA Security Service

The CORBA Security Service (CORBAsec) specification [8] has been developed by OMG to allow for implementation of secure, distributed CORBA-based systems. CORBAsec is one of the most extensive CORBA standards and specifies a security model and its architecture, as well as security facilities and interfaces for developers and administrators. Additionally, it describes how CORBAsec influences and changes the communication protocol between ORBs. Since secure services affect other object implementations, CORBA services and the ORB itself, the specification describes rules, which an implementation has to conform to, in order to behave as a CORBA security service.

The CORBAsec standard distinguishes between security-unaware (security level 1) and security-aware (security level 2) applications. Due to the fact that security-unaware applications were developed without security in mind, it is required that for that class of applications the ORB transparently utilizes the CORBAsec security mechanisms. On the other hand, a security-aware application is able to fully exploit the API (Application Programmer's Interface) of a CORBAsec implementation, i.e. it could implement custom mechanisms for authentication. One may notice that in the latter case security settings, which have to be propagated to client applications, are visible only in an application's source code (as a series of API calls) – this is contradictory to object-oriented concepts such as interfaces and data encapsulation.

CORBAsec's API relies on cryptographic techniques, which provide the actual security mechanisms. These mechanisms are not exposed to application objects via interfaces ensuring CORBAsec's application independence. CORBAsec implementations can be based on various security infrastructures, such as Kerberos, the Distributed Computing Environment (DCE), Sesame or Secure Socket Layer (SSL).

The general problem with CORBAsec is the granularity of access control. The smallest unit of control is an interface, which makes it impossible to vary the degree of security for different implementations of the same interface. Moreover, different instances of an interface implementation have to have the same degree of security. On the other hand, CORBAsec provides the concept of domains. A domain consists of several objects, which could implement different interfaces, and a set of access restrictions. Using the domain concept, different degrees of security for several objects implementing the same interface could be achieved by grouping them in small, different domains. However, this results in a significant administrative overhead.

Following the concept of role-based security, we have developed an extension to CORBAsec, which allows security control on the level of individual objects. Furthermore, we have developed an XML-based security description language, which is independent from the actual object implementation.

2.2 CONTROL - Access Control for ORBAsec

CONTROL [4] enhances the secure Object Request Broker ORBAsec SL2 developed by Adiron. ORBASec was the first publicly available CORBA implementation providing security services, which at least partially conform to CORBASEC.

CONTROL enables ORBAsec to provide access control to security-unaware applications, with only small changes to the code. Conforming to CORBAsec, access control is based on interface types.

Access control in CONTROL is describes via SAL, the *Server Access Language*. Figure 1 shows an interface and the corresponding SAL description.

```
module test{     (Credentials isBart
interface Hello{ (AccessId "bart@simpson"))
  void hi();
  void hello();  (CredentialsControl
  };              OnlyBart
};                  ((isBart Allow)
                     (true Disallow)))

                 (OperationControl
                  HelloBartOnly
                    "IDL:/test/Hello:1.0"
                   (("hi"   OnlyBart)
                    ("hello"((true Allow)))
                 ))
```

Figure 1: Interface and SAL description

For the interface `IDL:/test/Hello:1.0`, access to the method `hi()` is granted to a *CredentialsControl* with the identifier `OnlyBart`, which in tun is defined as a list of credentials. A credential is the unit, which access is granted or not. In our example, access to method `hi()` is only granted to a credential with *AccessId* `bart@simpson`. Access to method `Hello()` is not restricted.

2.3 A View-based Control Model for CORBA

```
view BaseView          view DerivedView :
   controls T {           BaseView {
   allow:                 allow:
      op_1;                  op_4;
      op_2;                  op_5;
   deny:               };
      final op_3;
      op_4;
};
```

Figure 2: View-based security – an example

In [1] Gerald Brose presents an alternative description model for access control based on views. Views are a new, object-oriented concept for the definition of access control. A view is defined by a set of operations on an interface. A view is then associated with an object, and specifies the access restrictions for that object. A principal is granted access to an object's methods, if the principal's view contains these methods. Views are extendable using view inheritance. Views are no IDL extensions. Changes to an object's security specification do not affect the IDL interface.

Figure 2 shows an example for view inheritance. Assume we have an interface `T` with methods op_1() to op_5(). The view `BaseView` allows access to methods op_1() and op_2() while it refuses it for `op_3()` and `op_4`. The keyword `final` is used to prevent derived views from changing the access specification of `op_3()`. `DerivedView` inherits the access specification from `BaseView`. In case of conflicting specifications, the specification in the derived view overrides the specifications from the base view. In the example, access to `op_4()` and `op_5()` is allowed.

The views concept is topic of ongoing research at Freie University Berlin. It will eventually be incorporated into a partial CORBAsec implementation for the public domain Java ORB JacORB [1] (which has been developed at FU Berlin as well).

3. Aspect-Description for Security

One of the main problems of incorporating security information into component-based distributed programs arises from the fact, that this security information usually has to be expressed through a dedicated API within the component's source code. The previously mentioned implementations of the CORBAsec level-2 specification are typical representatives of this practice.

We have developed a security description language, which introduces a simple syntax to express access control policies for CORBA objects – independent from the objects' implementations. Key components of our description technique are the definition of roles and the specification of per object-instance access rights.

Our security description technique has been implemented based on the ORBacus Object Request Broker (ORB) [10] and the JFSSL extension. In our implementation, per object access rights and authentication data are contained in security description documents. The syntax of those documents is based on a separate Document Type Description (DTD) for the eXtended Markup Language (XML). Our DTD for role-based security consists of three separate portions, namely "role and user description", "access control description", and "authentication description". These portions are now described in some detail.

3.1 Role and User Description

Many organizations and institutions grant access rights and privileges to their employees based on a role-concept. The organizational structure of those institutions usually is reflected by a set of roles (often organized as trees), which group users and extend rights. Our approach to security description closely follows this idea and supports the definition of multiple role-trees. Within a tree, a child-node represents a specialized role of its parent node. Therefore, a child node inherits access rights granted to its parent node (role).

Figure 3 shows the fragment of a Document Type Description (DTD) for role hierarchies in the above-mentioned tree structure.

```
01 <!ELEMENT role-definition (role+)>
02
03 <!ELEMENT role EMPTY>
04 <!ATTLIST role name ID #REQUIRED>
05 <!ATTLIST role parent IDREF #IMPLIED>
```

Figure 3: Role definition – DTD for security

Let us discuss the DTD shown in Figure 3 in some detail: The “role-definition”-entity shown in line 1 is the top symbol of a role definition and contains at least one “role”-entity (as indicated by the “+” sign). The “role”-entity (line 3) defines a single access role. It carries two attributes, a mandatory, globally unique identifier (line 4) and an optional reference on a parent role-identifier (line 5). If a reference to a parent role is given, the definition of the parent role has to be part of the same security description document.

```
01 <!ELEMENT userlist (user+)>
02
03 <!ELEMENT user (role-id+)>
04 <!ATTLIST user id ID #REQUIRED>
05 <!ATTLIST user description CDATA #IM-
PLIED>
06 <!ATTLIST user organization CDATA #IM-
PLIED>
07 <!ATTLIST user orgunit CDATA #IMPLIED>
08 <!ATTLIST user country CDATA #IMPLIED>
09
10 <!ELEMENT role-id EMPTY>
11 <!ATTLIST role-id name IDREF #REQUIRED>
```

Figure 4: User/security role association

The syntactical constructs necessary to express the association between users and roles are presented in Figure 4. The “userlist”-entity (line 1) is the top-level symbol. It has to be followed by at least one “user”-entity. A “user”-entity (line 3-8) is described by a globally unique identifier, followed by an optional description, which is constituted from “description”, “organization”, “orgunit” and “country” fields.

A „user“-entity has to contain at least one „role-id“-entity (line 10 and 11), which references a previously defined role (see Figure 4). The notion of “user” refers to a security principal (actual user or software component acting on behalf of the user).

3.2 Access Control Description

```
01 <!ELEMENT accesscontrol
          (iorobject|namedobject)+>
02
03 <!ELEMENT iorobject (method+)>
04 <!ATTLIST iorobject iorfile CDATA
05 <!ATTLIST iorobject interface CDATA
06 <!ATTLIST iorobject name ID #REQUIRED>
07 <!ELEMENT namedobject (name, method+)>
08 <!ATTLIST namedobject interface CDATA
09   #REQUIRED>
10 <!ATTLIST namedobject name ID
11   #REQUIRED>
12 <!ELEMENT name (part+)>
13
14 <!ELEMENT part EMPTY>
15 <!ATTLIST part id CDATA #REQUIRED>
16
17 <!ELEMENT method (accessrole+)>
18 <!ATTLIST method name CDATA #REQUIRED>
19
20 <!ELEMENT accessrole EMPTY>
21 <!ATTLIST accessrole id IDREF
     #REQUIRED>
```

Figure 5: Access control policy – DTD for security

In order to describe access control policies on the level of objects instances, a non-ambiguous mapping between objects and access control policies has to be established. The CORBA Interoperable Object Reference (IOR) provides the basis for such a mapping. A server object’s IOR can be obtained either from the CORBA naming service (once the service is registered) or directly from the server (e.g. via calls to object_to_string()/ string_to_object()). A common practice is to use a file (with well known name) to publish a server object’s IOR. The association between security role description (as expressed in the XML document) and actual CORBA object is based on the name of the file containing the IOR or on the corresponding entry

to the CORBA naming service. Figure 5 presents the part of the XML-Document Type Description necessary to describe an access control policy.

As shown in Figure 5, an access control policy consists of a mixed series of “iorobject”-entities and “namedobject”-entities (see line 1). The “iorobjects”-entity requires as attribute the name of the file containing the object's IOR (line 4), an optional identifier for the object's IDL interface (line 5) and a descriptive identifier for the object (line 6).

The "namedobject"-entity requires another entity as argument, which reflects the structured naming context used by the CORBA naming service to reference the object (line 8). The fine-grained description of this name is given in a number of "part"-entities (line 14 and 15). Subsequent attributes are similar to those given for "iorobject"-entities.

Both types of entities – "iorobject" and "namedobject" – are defined by a sequence of "method"-entities. Each "method"-entity takes an attribute referring to the name of a method. Since CORBA IDL does not use method overloading, it is sufficient to distinguish method names (and ignore formal parameter lists). A "method"-entity has to contain at least one "accessrole"-entity, which has to be defined in the same security description document.

```
01 <!ELEMENT accesscontrol
   (iorobject|namedobject)+>
02
03 <!ELEMENT iorobject (method+)>
04 <!ATTLIST iorobject iorfile CDATA
    #REQUIRED>
05 <!ATTLIST iorobject interface CDATA
    #REQUIRED>
06 <!ATTLIST iorobject name ID #REQUIRED>
07
08 <!ELEMENT namedobject (name, method+)>
09 <!ATTLIST namedobject interface CDATA
    #REQUIRED>
10 <!ATTLIST namedobject name ID #REQUIRED>
11
12 <!ELEMENT name (part+)>
13
14 <!ELEMENT part EMPTY>
15 <!ATTLIST part id CDATA #REQUIRED>
16
17 <!ELEMENT method (accessrole+)>
18 <!ATTLIST method name CDATA #REQUIRED>
19
20 <!ELEMENT accessrole EMPTY>
21 <!ATTLIST accessrole id IDREF #REQUIRED>
```

Figure 6: Authentication data – DTD for security

The mechanisms described up to this point work well for a priori known, static objects. Many CORBA applications heavily rely on such static objects, whose configuration remains unchanged after an initial setup phase. Within our framework, the programmer may configure access rights for a component based on the static objects exposed by the component.

Object factories are another important design pattern within the context of component-based software systems. However, in contrast to static, a priori known objects, it is hardly possible to describe individual access control policies for dynamically created objects without inserting explicit security calls into the calling object's source code. In fact, CORBAsec allows specification of access rights on the (static and class specific) interface level only.

We propose and have implemented the following technique for dealing with dynamically created CORBA objects: Each security principal (user) is assigned an implicit personal role which states full access rights for all objects dynamically created on behalf of the principal. From this point of view, a call to an object factory is nothing else but creation of an anonymous object – the returned object is accessible only by the security principal who initiated the call.

3.3 Authentication Description

The previous sections have described how access rights can be defined based on object identity and corresponding role definition. However, since CORBA is acting in insecure, distributed environments, it is necessary to establish proof of identity between security principals prior to any security checks. Our implementation of aspect-based security uses the JFSSL extension to ORBacus as implementation platform. JFSSL is an implementation of the Secure Socket Layer (SSL) protocol, which uses certificates to identify security principals.

Our XML Document Type Definition contains a subsection, which allows specification of an identity for previously defined CORBA objects. Multiple objects can be grouped into a server component, which an SSL certificate is assigned to. Figure 6 shows how authentication-related data (SSL-certificates) for a component can be specified. However, our framework is open for other authentication techniques as well.

4. Tele-Lab: a Distributed Control System

In a collaborative effort with the University of Illinois Urbana-Champaign (UIUC), we have developed a distributed tele-laboratory application, which utilizes Simplex [12], a fault-tolerant real-time control architecture previously developed at UIUC.

Experimentation with real physical systems is a key part in control education. Traditionally, the use of real physical systems required students' physical presence in the laboratory. Our Tele-laboratory project allows students to remotely conduct physical experiments via the Web easily and reliably, creating an integrated virtual and real laboratory education environment. A simple time-sharing scheme ensures better usage of the equipment.

Using the Tele-laboratory, students may access equipment (control devices such as an inverted pendulum or a robot) over the Web without having to travel to a physically remote location in order to carry out experiments. Using the software, students can write a controller for a device, upload it to the controlling machine and try it out on physical equipment with no possibility of damaging the device.

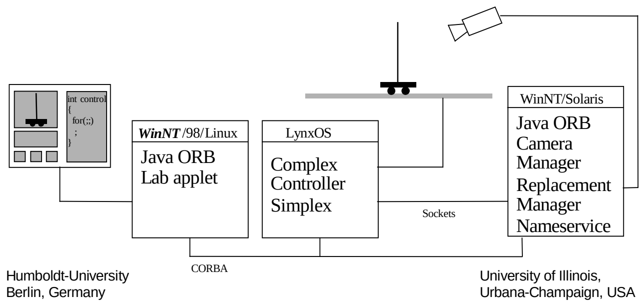

Figure 7: Tele-laboratory scenario

In an instructional situation it is likely that the user-supplied controllers will malfunction, possibly in dangerous ways. Due to the reality of these dangerous malfunctions, it is necessary to take precautions to protect non-fail-safe laboratory equipment. Simplex provides this protection using the technique of analytic redundancy, and can detect and replace a malfunctioning controller with the predefined safety controller to keep the equipment safe and in working order. The students is given feedback as to the state of their controllers (running, failed), as well as the system states that led to the termination of their controllers in the case of failure. In addition, the students are provided with a video stream of the physical device and graphs of its status. Facilities are also in place for the controller to transmit debugging information output during a run back to the student after execution completes (so as to not interfere with the real-time requirements of the student's controller).

The scenario depicted in Figure 7 shows the inverted pendulum controlled by Simplex running on a LynxOS 3.0 system, along with the components of the Tele-laboratory system. Using our "Composite Objects" technology [11] , the replacement manager encapsulates the user-interface of simplex and exports this functionality to CORBA objects. CORBA objects can request the compilation of controller code by communicating with the replacement manager, which also has an interface to the C compiler. We have used ORBacus 3.2 for Java and C++ for interfacing Simplex and the tele-laboratory components.

The lab applet is the user-visible portion of the Tele-laboratory. Students use this application to load or develop controller code, for communicating with the replacement manager and watching the video and execution status feedback. The application provides quality of service choices to the student, in the form of multiple video streams of different bandwidth.

The actual scenario shown in Figure 7 relates to an experimental setting where the lab applet has been run on a computer at Humboldt-University of Berlin, Germany, whereas the Simplex control application was executed at University of Illinois, Urbana-Champaign. A Logitech Web camera has been used to capture the experiment's status. The WebVideo system from University of Ulm, Germany [13] has been set up to prepare and transmit the camera's video stream via the internet.

Reliable operation and online-component replacement were the main goals of the initial version of tele-laboratory. Security issues were only of minor concern. In order to maintain minimal security, ports and names of crucial components were simply hidden. However, since system reliability is affected by security attacks as well, a stronger and more flexible security concept was needed for a production version of the tele-laboratory.

5. Security Control for Tele-Lab

5.1 Security description

Since communication inside the tele-laboratory is centered around a CORBA event channel, securing the event channel is crucial. Figure 8 shows the interface of an event channel as defined in the CORBA standard. It provides methods for registration with the event channel to event producers and consumers. Following the factory pattern, the methods dynamically create new objects (`ConsumerAdmin` and `SupplierAdmin`). Our security model implicitly controls access to those objects and grants access to them only to the user who created them (as described in Section **Error! Reference source not found.**). Therefore, only access rights to the top level factory have to be specified, which, in our case, is the event channel interface itself.

```
interface EventChannel {
    ConsumerAdmin for_consumers();
    SupplierAdmin for_suppliers();
    void destroy();
};
```

Figure 8: CORBA EventChannel interface

A user of the tele-laboratory employs the lab applet in order to remotely carry out an experiment. During a session, the lab applet generates various CORBA events, which are sent to the backend over the event channel. Therefore, security can be achieved by controlling access to the `for_suppliers()` method of the event channel interface. Every object, which wants to send events over the event channel needs access to this method. In order to prevent snooping by unauthorized objects, access to `for_consumers()` has also to be controlled.

```
<role-definition>
  <role name="Student" />
  <role name="RemoteLabAdmin"
        parent="Student"/>
</role-definition>

<userlist>
  <user id="Mueller" >
    <role-id name="Student"/>
  </user>
  <user id="Meier" >
    <role-id name = "Student" />
  </user>
  <user id="Schmidt" >
    <role-id name="RemoteLabAdmin"/>
  </user>
</userlist>
```

Figure 9: Role definition for access to tele-lab

The method `destroy()` is called for garbage collection on unused event channel proxies. Controlling access to this method ensures that the event channel cannot be deleted by an unauthorized object.

```
<accesscontrol>
  <iorobject iorfile="/path/to/ior"
             inter-
face="IDL:omg.org.CosEventChannelAdmin.
                        EventChannel:1.0"
             name="EventChannelAdmin" >
    <method name="for_consumers" >
      <accessrole id="Student" />
    </method>
    <method name="for_suppliers" >
      <accessrole id="Student" />
    </method>
    <method name="destroy" >
      <accessrole id="RemoteLabAdmin" />
    </method>
  </iorobject>
</accesscontrol>
```

Figure 10: Access rights for tele-lab's event channel

Figure 9 shows the definitions for authorized users and their associated roles. It is based on the DTD shown in Section 3.1. The description defines two roles, Student and RemoteLabAdmin, which allow to distinguish between users of the tele-laboratory and it's administrator. Besides being allowed to administrate the tele-laboratory, a RemoteLabAdmin can also use it, since he inherits from Student. The user list contains the certificate ids Mueller and Meier, who are allowed to access the tele-laboratory as students, and the administrator Schmidt.

A valid access control configuration for the tele-laboratory's event channel is shown in Figure 10. The event channel's IOR was saved in a file and therefore the tag `iorobject` has to be used. Using the attributes, the path to the file, the interface name, and a unique name are specified.

Access to that object is defined as explained above. `for_consumers()` and `for_suppliers()` can be called by `Student`s only, while only `RemoteLabAdmin` is allowed to call `destroy()`.

5.2 CORBA Integration – Implementation Issues

In order to use our description technique for role-based security in conjunction with a CORBA implementation, the underlying ORB has to provide sufficient security functionality. As mentioned in Section 2 only a very limited number of CORBA implementations provide that functionality. Therefore, we have extended the public domain CORBA implementation ORBacus 3.3.1 in such a fashion, that it accepts XML documents adhering to our security DTD. We have used the secure socket layer (SSL) extension JFSSL for ORBacus, which provides an authentication API. Access control mechanisms have been implemented within ORBacus interceptors (Unfortunately, the implementation of interceptors in ORBacus is not yet compliant to the CORBA specification).

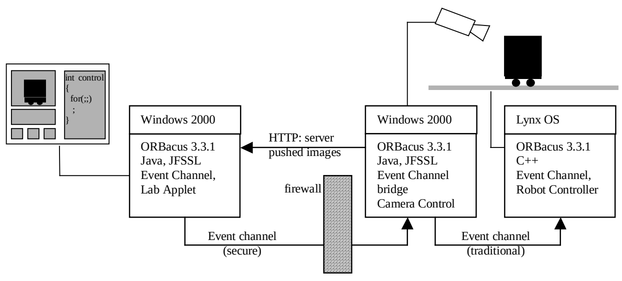

Figure 11: Security-enhanced Tele-lab scenario

Additionally, our extension to ORBacus supports a remote administration interface, which allows to change the currently effective security policy for the ORB. This interface is used by the Security Administration Manager tool, which also provides a convenient method to create and manage role descriptions, access rights, and authentication settings.

5.3 CORBA Integration - pitfalls

Using our programming libraries for the security-enhanced ORBacus, it is possible, to implement access control for a Java-based CORBA application without any changes to the application's source code. As a proof-of-concept application we have implemented access control for the tele-laboratory. Users of the lab-applect may obtain personnel SSL certificates which they may use for authentication. The only change necessary is a adjustment to the Java classpath for both, CORBA client and server.

Unfortunately, the tele-laboratory backend (which interfaces to the Simplex real-time control software) had to be implemented using a C++-CORBA implementation on the LynxOS real-time operating system, which does not interoperate with the Java-based SSL implementation (JFSSL). Since all parties of an event channels have to use the same underlying transport, we had to refine the overall communication structure of the tele-laboratory. We have introduced an event channel bridge as additional component. On server-side, located behind a firewall, this bridge negotiates between a secure event channel visible to the outside world and a traditional internal event channel, which is accessible to the legacy Simplex control system. Figure 11 shows the communication structure of our security-enhanced version of the tele-laboratory.

6. Conclusions

With the increasing number of Web-based distributed applications, quality-of-service properties such as reliability, predictability, and security are of an essence not only to special purpose control systems, but to the majority of web users. State-of-the-art technologies for distributed component-based (Web) applications, such as CORBA and Java support the description of components' functional interfaces. However, in contrast to interface definition languages (such as CORBA IDL), which describe a component's functional interface, there is no general means to describe a component's non-functional properties, such as security settings, fault-tolerance measures and timing behavior.

Within this paper, we have presented our approach for role-based security for CORBA. Following the idea of aspect-oriented programming, we have developed tools and a description language for security settings. The description language uses the eXtended Markup Language (XML) as an underlying representation and allows specification of access rights independently from an object's implementation. We have discussed the necessary description and configuration steps for a secure CORBA service. We have demonstrated how our previously developed distributed tele-laboratory application can be configured for secure access. Most notably, introduction of those security features did not require changes to the application's source code. Our implementation of role-based security for CORBA uses the ORBacus 3.3.1 ORB and the JFSSL secure socket layer library as foundation.

References

[1] Gerald Brose; "JacORB – design and implementation of a Java ORB"; Distributed Applications and Interoperable Systems, Chapman and Hall, 1997.

[2] Gerald Brose; "A View-Based Access Control Model for CORBA"; in Secure Internet Programming: Security Issues for Mobile and Distributed Objects, Springer, 1999.

[3] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn; "Role-based Access Control"; Proc. Of 15th National Computer Security Conference, Gaithersburg, Maryland, Springer Verlag, 1992.

[4] Polar Humenn; "A Language for Access Control in CORBA Security"; Technical Report Adiron, Syracuse University, Syracuse, NY, 1999.

[5] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin; "Aspect-Oriented Programming", In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241, June 1997.

[6] OMG; "Fault tolerant CORBA Using Entity Redundancy RfP"; OMG Document orbos/98-04-01, at http://www.omg.org.

[7] OMG, "The Common Object Request Broker: Architecture and Specification. Revision 2.3", OMG (Object Management Group) document, ftp://ftp.omg.org/pub/docs/formal/98-12-01.ps.

[8] OMG, "CORBAsec: Security Service, version 1.5", OMG (Object Management Group) document, ftp://ftp.omg.org/pub/docs/formal/2000-06-25.ps.

[10] Object-Oriented Concepts, Inc; ORBacus Documents and Sources, http://www.ooc.com/ob/download.html.

[11] Andreas Polze, Lui Sha; "Composite Objects: Real-Time Programming with CORBA"; in Proceedings of 24th Euromicro Conference, Network Computing Workshop, Vol.II, pp.: 997-1004, ISBN 0-8186-8646-4, Vaesteras, Sweden, August 25-27, 1998.

[12] L.Sha, R.Rajkumar, M.Gagliardi; "Evolving Dependable Real-Time Systems"; in Proceedings of 1996 IEEE Aerospace Applications Conference, IEEE Inc., February 1996, ISBN 0-7803-3196-6.

[13] Heiner Wolf, Konrad Froitzheim; "WebVideo – A tool for WWW-base Tele-cooperation"; in Proceedings of IEEE International Symposium for Industrial Electronics (ISIE), Guimaraes, Portugal, IEEE Inc., July 1997