

Deliverable AP 3.9

# Tool Design for Automated Legacy System Policy Analysis

Subject Area ENFORCE

14.02.2008

Mathias Kohler, SAP AG  
Christian Wolter, SAP AG

ORKA is funded by the German Ministry of Education and Research (BMBF) as part of its Software Engineering 2006 programme.

© 2007 ORKA Consortium



Federal Ministry  
of Education  
and Research

Internal document information:

\$Id: del-ap3.9.tex 903 2008-02-15 10:02:37Z kohler \$

# Contents

- 1 Introduction 4**
- 2 Legacy System and Technology Analysis 4**
  - 2.1 SAP Web Application Server . . . . . 4
    - 2.1.1 User Management Engine . . . . . 5
    - 2.1.2 Policy Model . . . . . 6
    - 2.1.3 Model Matching . . . . . 7
    - 2.1.4 Interfaces . . . . . 9
  - 2.2 Sun Java Access Manager . . . . . 11
    - 2.2.1 Policy Model . . . . . 11
    - 2.2.2 Model Matching . . . . . 13
    - 2.2.3 Interfaces . . . . . 15
  - 2.3 Active Directory . . . . . 17
    - 2.3.1 Policy Model . . . . . 17
    - 2.3.2 Model Matching . . . . . 17
    - 2.3.3 Interfaces . . . . . 18
  - 2.4 JBossSX - JAAS Security . . . . . 19
    - 2.4.1 Policy Model . . . . . 21
    - 2.4.2 Model Matching . . . . . 23
    - 2.4.3 Interfaces . . . . . 24
  - 2.5 Oracle Access Manager . . . . . 26
    - 2.5.1 Policy Model . . . . . 26
    - 2.5.2 Model Matching . . . . . 29
    - 2.5.3 Interfaces . . . . . 30
  - 2.6 IBM Tivoli Access Manager . . . . . 31
    - 2.6.1 Policy Model . . . . . 31
    - 2.6.2 Model Matching . . . . . 34
    - 2.6.3 Interfaces . . . . . 34
- 3 Legacy Adapter Tool 38**
  - 3.1 Design Concepts . . . . . 38
  - 3.2 Integration Scenario . . . . . 39
  - 3.3 API Definition . . . . . 40
- 4 Conclusion 44**

# 1 Introduction

This work package in the context area of *Enforce* deals with a detailed analysis of some typical and popular enterprise legacy applications. In order to migrate existing system setups into an ORKA enhanced system landscape.

The ORKA control architectures is designed to support various system landscapes and enterprise applications. Due to the unique features of ORKS's security policy expression language OPL, authorisation policies based on different access control models and concepts, such as role-based access control, task-based access control, team-based access control, time-based access control, or the most generic one, attribute-based access control model, are supported by OPL.

In the case the ORKA architecture is introduced into an enterprise-scale system environment old security configurations must be ported and translated into OPL policies. Because it is unlikely that system administrators want to recreate all enterprise-wide security configuration from scratch we investigate in the course of this work package some popular and wide-spread legacy applications and analyse how far existing system configurations can be migrated and translated into the ORKA architecture and its OLP policies. Six representative legacy applications and their utilized security technology and authorisation models are discussed and mapped onto the OPL model of ORKA.

The rest of this document is structured as follows. In section 2 we give an overview and analysis of six different legacy applications and their security modules, namely SAP Web Application Server, SAP BusinessByDesign, Microsoft's Active Directory, JBossSX, Oracle Access Manager, and general SOA security frameworks, implementing the WS-Policy and WS-Security standard. In section 3 we provide an architectural conception of an legacy policy migration tool that is able to connect to these selected legacy applications, queries parts of their security configuration and translates them into OPL policies that can be deployed in an ORKA environment. This work package concludes with a discussion about the implications of the legacy system analysis for the prototype implementation of the upcoming work package 3.10

## 2 Legacy System and Technology Analysis

In this section we will give a detailed description six different legacy applications that are common in an enterprise-scale system landscape.

### 2.1 SAP Web Application Server

The SAP Web Application Sever (AS) is the major component of the SAP NetWeaver stack. It represents the application platform for all Java and ABAP-based SAP applications, for instance the SAP Portal. The AS enabled the migration from former monolith SAP applications into service-oriented applications deployable in an enterprise service-oriented architecture.

While the Web AS is supporting two types of authorisation concepts for Java-based applications, such as Java Authentication and authorisation Service (JAAS) and the SAP User Management Engine (UME), we will focus in this section on the UME and the AS-Java stack. We will

describe the JASS concepts in the context of the JBoss application server in one of the following sections that apply for both SAP AS-Java and JBossSX. Authorisation concepts for the SAP AS-ABAP have been discussed already in work package 3.1. and will not be repeated in this section. Parts of this documentation are taken from [SAP05].

### 2.1.1 User Management Engine

User Management within the SAP Web Application Server is performed by the User Management Engine (UME). UME user data is stored in one or more data sources. Each type of data source has its own persistence adapter in the UME. The persistence manager consults the persistence adapters when creating, reading, writing, and searching user management data (cf. Figure 1). Three different types of persistence adapters are supported by SAP AS:

- Database, such as MaxDB
- Lightweight Directory Access Protocol (LDAP) directory
- SAP Systems based on Web Application Server 6.20

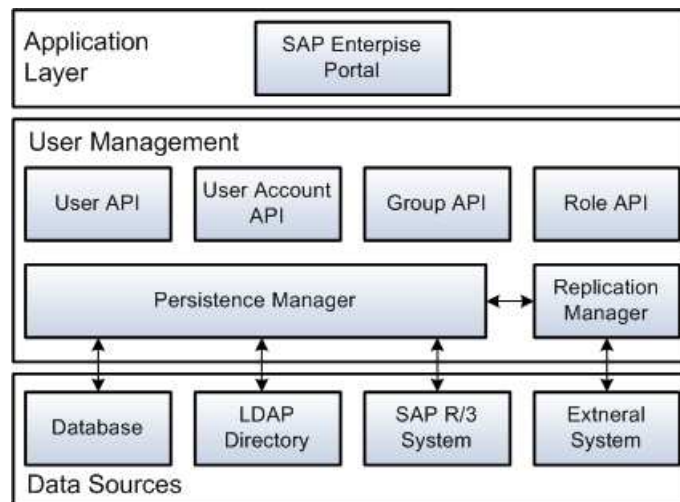


Figure 1: SAP R/3 User-Management Layers

The data repositories or persistence layers from which the user management engine (UME) retrieves user management data are referred to as data sources. With the UME, the leverage of existing user data repositories to the UME system infrastructure by connecting to it using configurable persistence adapters (cf. Figure 1).

The user management configuration can be used to define and setup the connection to the various user data sources, such as indicated by the screen shot in Figure 2. It is possible to setup several data stores in parallel. Users can also be stored in several different physical LDAP directory servers, or in different branches of the same LDAP directory server. It is also possible to configure UME so that user data is read from an existing corporate directory, while new users are written to the SAP NetWeaver Application Server (AS) Java database.

A persistence manager is responsible for reading the data from or writing the data to the correct data source. The data source to which the persistence manager writes is transparent to applications using UME.

The configuration of the UME for the different data source types is defined by the data source configuration file. The data source configuration file is an XML file that defines a configuration for standard scenarios, such as storing standard user data in a corporate LDAP directory (directory service) and application-specific data in the AS Java database.

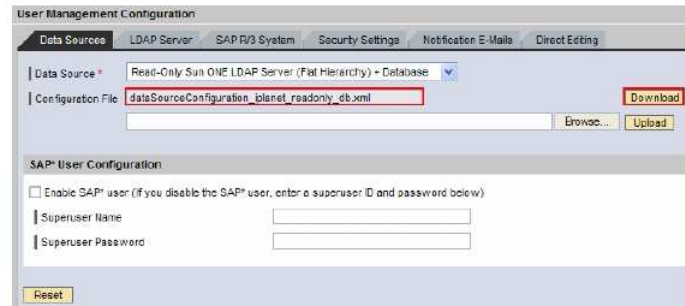


Figure 2: User Management Configuration

The data source(s) must be available for the AS to start. The AS is dependent on authentication and authorisation information from the data source. If the UME is misconfigured or the data source is not available on the network, the SAP AS can not start. The UME can use the following types of data sources:

- Database of the AS Java
- Directory service
- User management of the AS ABAP

## 2.1.2 Policy Model

Authorisations are enforced in the user management engine (UME) using permissions, actions, and roles. Internally in their code, applications define UME permissions and use them for access control. UME permissions are an implementation of Java permissions.

An action is a collection of permissions. Every application defines its own set of actions and specifies the permissions assigned to the actions either in an XML file or dynamically in the code. The actions appear in the user management administration console, where they can be grouped together into roles. UME Roles group together actions from one or more applications. Roles are assigned to users or groups of users in the user management administration console to define user authorisations. There exist three different types of XML-file based specification of permissions:

- **Java Security Roles**

The application developer defines and deploys security roles together with the J2EE application he created in accordance with the J2EE specification. The deployment descriptors for the role are included in the WAR file for Web modules or the JAR file for EJB modules.

- **UME Roles**

UME consist of actions which are in turn collections of permissions used for Web Dynpro applications. UME actions are deployed with an application and defined in a file called *actions.xml*.

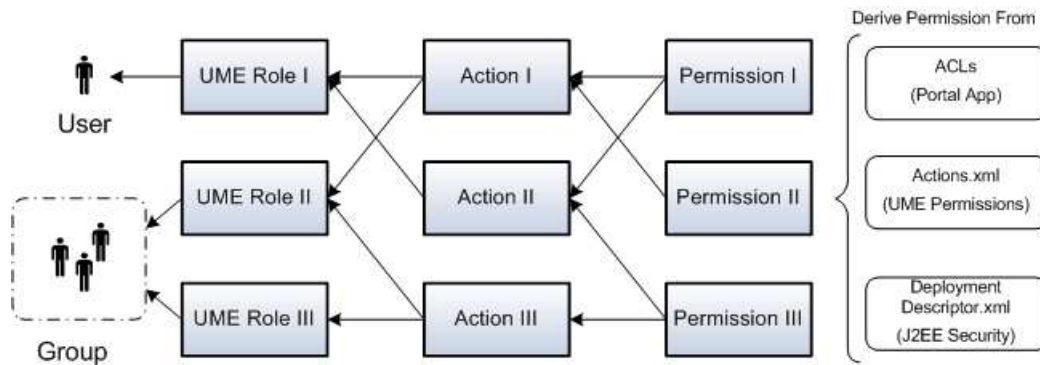


Figure 3: UME Role Concept

- **Access Control Lists**

ACLs limit access to individual objects. The portal is one application that uses ACLs to control access to objects on the AS Java. One example of this is the portal content directory (PCD). The UME also provides APIs for maintaining ACLs on the AS Java.

### 2.1.3 Model Matching

In this subsection we describe the most important business objects of SAP AS and match them with related entities of the OPL model. Therefore, we will list the related business objects:

- **User**

The most important object used by the identity management APIs is the business object USER. User contains the technical information used for logon and work in the SAP system, for example, validity, role and profile assignments and printer settings.

- **Technical User**

Technical user are created for system to system communication. In most cases applications create their own users for communication automatically.

- **User Attributes**

User attributes contains various communication data, such as telephone number, fax, e-mail address, company assignment, and additional address data for each user.

- **Group**

The group business object is used by the identity management to is used to cluster individual USERS. There exists three default groups: *anonymous*, *authenticated*, *administrators*, *guests*, and *everyone*. Groups serve to create sets of users who have something in common, for example, users who work in the same department or users who have similar tasks in a company. Groups make it easier to manage users. For example, it is possible to assign a role to a group instead of assigning the role to each user individually. This is known as indirect role assignment.

- **Virtual Group**

Virtual groups are used to automatically assign users to User Management Engine (UME) groups based on the value or values of a single USER attribute. In the UME properties it is defined which attribute to use and which values are used to form groups.

- UME Role  
These roles define a set of authorisations. By assigning a user or group to a UME role, the set of authorisations that the role defines to the assigned user or group are granted.
- Portal Roles  
These roles define how content is grouped together and how it is displayed in the SAP NetWeaver Portal. By assigning a user or group to a portal role, it is defined which content that user or group sees in the portal. During assignment, the system checks the Role Assigner permission to see if the administrator has the proper rights to assign the role.
- Permission  
UME permissions are provided by the User Management Engine (UME) and extend the possibilities provided by the J2EE security roles. However, they are only supported when using a programmatic approach.  
  
One advantage for using UME permissions are that the administration is easier because it consolidates permissions into actions. The administrator works with the actions and not with all of the individual permissions.
- Action  
Actions are a consolidated collection of Permissions.

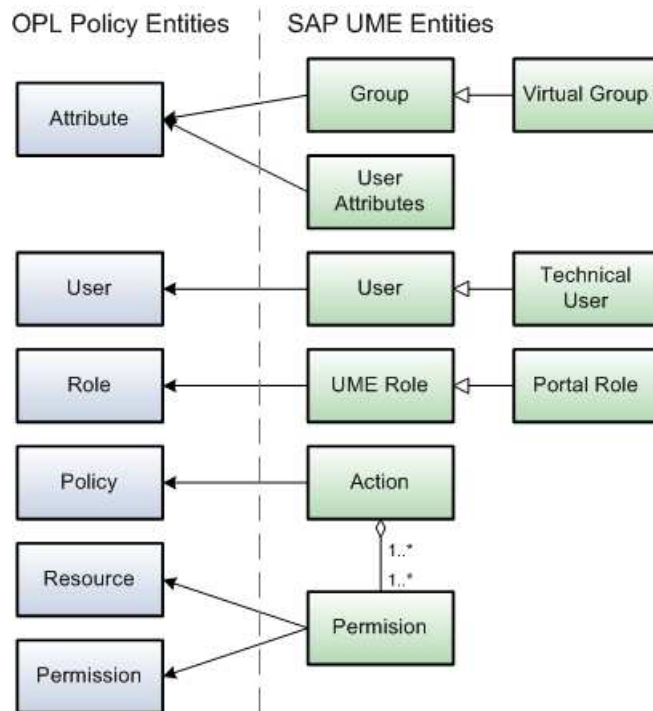


Figure 4: OPL-SAP-UME-Mapping

Referring to Figure 4 a mapping between OPL entities and SAP UME Entities is straight forward. Groups and groups based on dynamic attributes are translated into OPL Attribute entities that are assigned to User entities. Users are mapped to their OPL equivalents. Similar UME and Portals roles are translated into OPL roles. UME Actions can be represented by an OPL policy. Each policy than contains the collection of UME Permissions. The resource specific information necessary for OPL resources are extracted from the UME permissions. The related granted activity that is assigned to the affected resources is translated into an OPL permission.

#### 2.1.4 Interfaces

This section describes how to use the identity management Java APIs for managing users, groups, and roles when using the User Management Engine (UME) for identity management with SAP systems.

The following functions can be performed on user, group and role objects using the identity management APIs:

- Creating objects
- Modifying objects
- Searching for objects
- Deleting objects

The following Figure 5 shows a complete list of potential BAPI calls to access UME data:

Besides the ABAP based access by calling the listed BAPI functions, it is also possible to access UME data by utilizing the JAVA APIs *SAP UME SPML*. The Java-based APIs relate to

<b>Purpose</b>	<b>RFC Function</b>	<b>BAPI Method</b>
Obtain a list of users	BAPI_USER_GETLIST	<i>USER.GetList()</i>
Obtain information about users	BAPI_USER_GET_DETAIL	<i>USER.GetDetail()</i>
Value help	BAPI_HELPVALUES_GET	<i>Helpvalues.GetList()</i>
Create users	BAPI_USER_CREATE1	<i>USER.Create1()</i>
Modify users	BAPI_USER_CHANGE	<i>USER.Change()</i>
Delete users	BAPI_USER_DELETE	<i>USER.Delete()</i>
Set initial passwords	BAPI_USER_CHANGE	<i>USER.Change()</i>
Set a productive password	SUSR_USER_CHANGE_PASSWORD_RFC	None
Lock users	BAPI_USER_LOCK	<i>USER.Lock()</i>
Unlock users	BAPI_USER_UNLOCK	<i>USER.Unlock()</i>
Obtain a list of companies	BAPI_HELPVALUES_GET	<i>Helpvalues.GetList()</i>
Obtain information about a company	BAPI_ADDRESSORG_DETAIL	<i>AddressOrg.FindDetail()</i>
Modify address information of a company	BAPI_ADDRESSORG_CHANGE	<i>AddressOrg.Update()</i>
Obtain a list of roles	PRGN_ROLE_GETLIST	None
List role assignments	BAPI_USER_GET_DETAIL	<i>USER.GetDetail()</i>
Assign roles	BAPI_USER_ACTGROUPS_ASSIGN	<i>USER.ActgroupsAssign()</i>
Delete role assignments	BAPI_USER_ACTGROUPS_DELETE	<i>USER.ActgroupsDelete()</i>
Obtain a list of profiles	BAPI_HELPVALUES_GET	<i>Helpvalues.GetList()</i>
List profile assignments	BAPI_USER_GET_DETAIL	<i>USER.GetDetail()</i>
Assign profiles	BAPI_USER_PROFILES_ASSIGN	<i>USER.ProfilesAssign()</i>
Delete profile assignments	BAPI_USER_PROFILES_DELETE	<i>USER.ProfilesDelete()</i>

Figure 5: BAPIs and Function Modules Used for Identity Management

the SPML<sup>1</sup> protocol, in principal a simple request/response protocol for CRUD-like activities on the UME data sets. Following this approach two major constraints must be considered:

- SAP role objects cannot be created or deleted
- Only certain ABAP attributes are supported.

## 2.2 Sun Java Access Manager

The Sun Access Manager 7.1 is a comprehensive authentication and authorisation framework to protect network resources. The idea is - as with many other access managers as well - to protect all types of information stored within a company. For the Sun Access Manager the typical scenario is that a user within the company or an external application wants to access information content located on an internal server. There will be an Access Manager policy agent which intercepts the call and directs the request to an Access Manager server.

For the evaluation of the requested call the Access Manager needs to authenticate the user and therefore asks the user for his/her credentials. The entered credentials are matched to the ones stored internally in one of the identity repositories available for the Sun Access Manager. The connection to such repositories are realized using authentication modules which are responsible to compare the received user information with the information stored in the repository. Available repository plug-ins are, for example,

- Active Directory (uses LDAP version 3 specification for accessing data),
- Certificate-based authentication,
- HTTP Basic (credential transportation for internal validation using the LDAP authentication module),
- LDAP module,
- verification via an external RADIUS-server, or
- OS-based authentication

to name a few.

If the user is authenticated the request is evaluated by the Access Manager policy agent according to the stored policies. There are different types of policies to identify, for example, for which groups access to a specific resource may be granted. A complete list of policy types will be given in the section below, Policy Model. The policy evaluation results in a grant or deny answer based on which access is permitted or denied.

The following picture depicts a general integration scenario (taken from [Mic08d]).

### 2.2.1 Policy Model

There are two policy types provided by the Access Manager

---

<sup>1</sup>Service Provisioning Markup Language

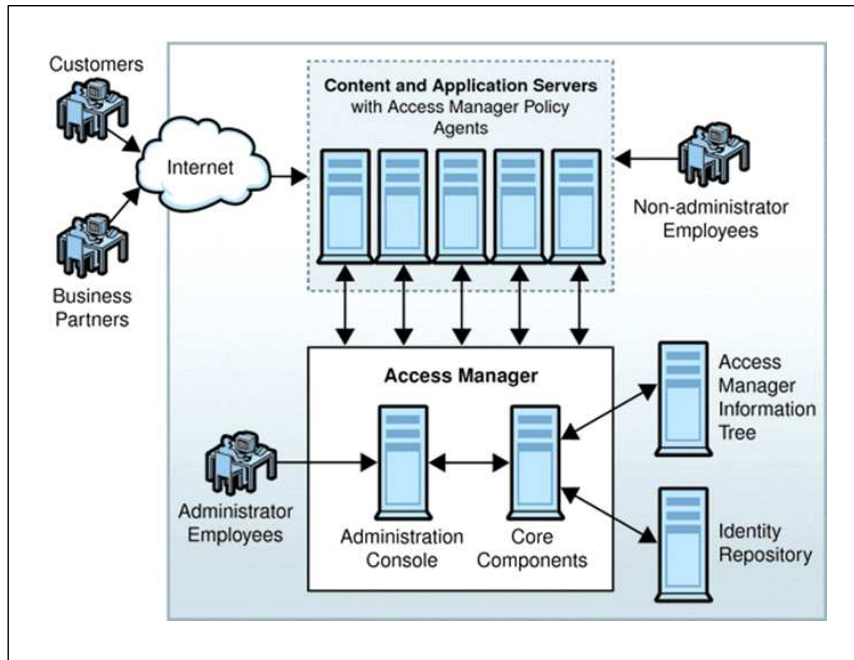


Figure 6: Sun Access Manager - General Integration Scenario [Mic08d]

- Normal Policy, and
- Referral Policy.

### Normal Policy

The "Normal Policy" specifies who can get permission to access a protected resource. It comprises rules, subjects, conditions, and response providers. Generally, the Policy is structured as XML document and therefore has the just mentioned parts as different elements. We will provide a brief description of each in the following paragraphs.<sup>2</sup>

The **Policy** element is the root element and may contain one or more of the mentioned elements Rule, Conditions, Subjects, Referrals, or Response Providers.

A **rule** defines a typical permission for a user to execute an action on a resource. It explicitly specifies the user, the resource, and the allowed actions on the resource by the elements Service-Name, and ResourceName. Further a "value" has to be defined which specifies the permission for the action, for example, "allow" or "deny".

The **subject** of a policy (or rules respectively) might be a single user or a collection of users, called group. Standard subjects which can be assigned to policies are, for instance, Access Manager Roles and Identities which are the subjects defined within the Sun Access Manager System, or LDAP Groups, Roles and Users. Additionally to the by standard available subject types, the Access Manager Policy API allows the use of customer specific implementations.

The **condition** part of a policy specifies additionally constraints which may result in restrictions for the user such as that access to a resource is only possible between a certain time frame. As with subjects Access Manager provides a standard range of possible conditions which also can

<sup>2</sup>The description is based on the Sun Access Manager Documentations [Mic08a], [Mic08c], and [Mic08d].

be extended using the Access Manager Policy APIs. As standard the main provided conditions are

- Active Session Time (e.g., restricts the maximum length of a user session)
- Authentication Chain
- Authentication Level ("The Authentication Level attribute indicates the level of trust for authentication. The policy is applicable [e.g.,] if the user's authentication level is greater than or equal to the Authentication Level set in the condition." [Mic08d])
- Authentication Module Instance (applies if the user has successfully authenticated with a specified authentication module)
- IP Address/DNS Name
- Current Session Properties
- Time (time, day, date, time zone)

The last element of a policy are the **response providers** which are plug-ins that provide means to deliver additional information about the user's profile in conjunction with a policy decision to the requesting application. There is one standard implementation available (IDResponse-Provider); others can be implemented using the Access Manager Policy API.

### **Referral Policy**

The second type of policies are called Referral Policies. They are meant for policy administration in such a way, that the Administrator at root level (or at the highest level of the Access Manager information tree) can delegate the creation of policies for resources to other administrators. By delegating the right to create policies, the users receiving the right may then also be able to create and configurate policies. A referral policy consists of rules (which define the resource) and the referrals (defining the identity objects to which the rights for policy creation and configuration is transferred). In summary, a referral policy is to delegate policy management privileges to other entities.

The Policy itself is finally stored XML-encoded in the Directory Server. The Policy Service loads the policies from there.

### **2.2.2 Model Matching**

The matching between the ORKA entities and the entities from the Sun Access Manager are straight forward. Resources entities referred to within Access Manager Policies can be directly mapped to the resource entities in ORKA. The Policies of the Access Manager consist of rules and conditions. The rules in a policy is the permission to access a resource based on direct user identities or roles or groups and a predefined action. This corresponds to the Permission entity with ORKA. The condition can be mapped to constraints with ORKA as they extend the notion of permissions by additional constraints which have to be fulfilled to be granted access to a resource. Finally user and group entities have to be mapped. Clearly, those map to the user and role entities on the ORKA side respectively. 7 depicts the mapping.

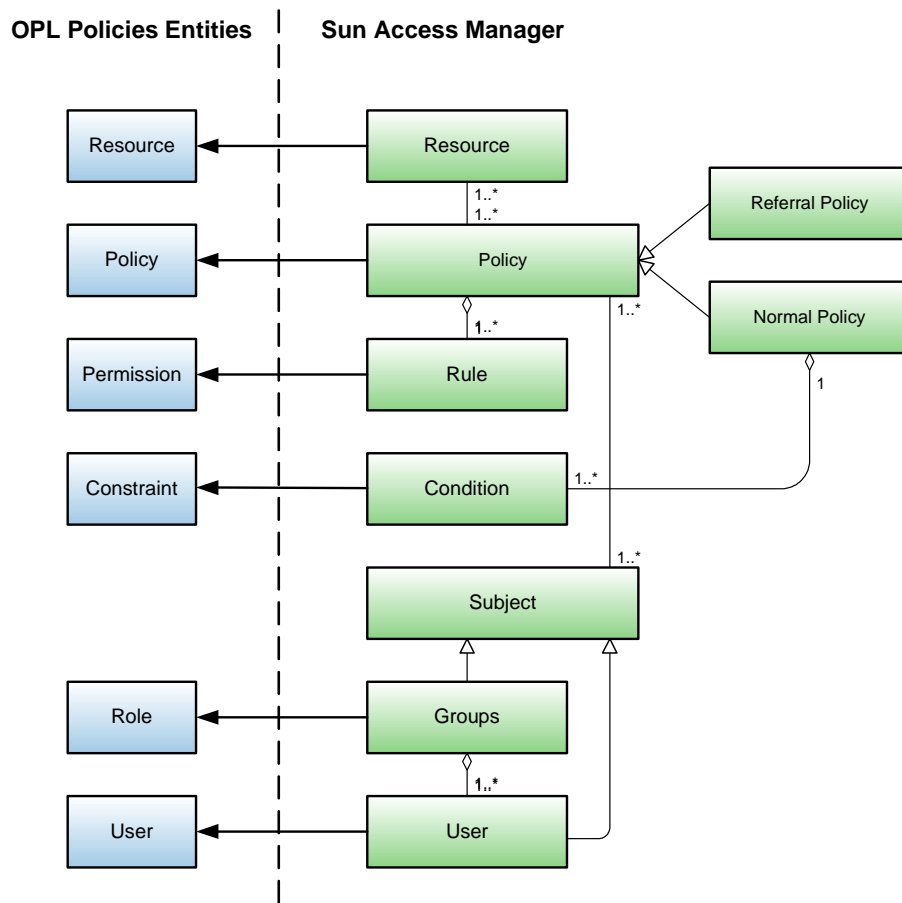


Figure 7: Mapping between Sun Access Manager entities and ORKA entities

Methods	Description
policyManager constructor	The PolicyManager can be obtained by passing a privileged user's session token or by passing a privileged user's session token with a realm name. The PolicyManager class manages policies for a specific organization, sub organization or a container.
Policy getPolicy(java.lang.String policyName)	Gets the policy object given the name of the policy.
java.util.Set getPolicyNames()	Gets a set of names of polices defined in the organization for which the policy manager was instantiated.
ReferralTypeManager getReferralTypeManager()	Returns ReferralTypeManager associated with this policy manager.
ResourceManager getResourceManager()	Gets the ResourceManager object instance associated with this PolicyManager object instance
SubjectTypeManager getSubjectTypeManager()	Gets the SubjectTypeManager object instance associated with this PolicyManager object instance

Table 1: Accessing the Policy Manager

### 2.2.3 Interfaces

Sun Microsystems provides for the Access Manager a Policy Application Interface (Policy API) and Policy Management Classes to configure the Access Manager itself (adding custom subjects, referrals, or conditions), as well as to create and manage policies located in the policy store. The API is provided in several languages, among them C and Java where we will centre on the latter.

As typical for Java the Policy API comes in different Java packages comprised by the following three *com.sun.identity.policy*, *com.sun.identity.policy.client*, and *com.sun.identity.policy.interfaces*. For Policy Management the Java packages *com.sun.identity.policy* can be used. For the gathering of the policies of an Access Manager System to transfer them to ORKA, the last package is of main interest and will be depicted in more detail in the following paragraphs.

The Java package *com.sun.identity.policy.PolicyManager* contains the classes and methods to access policies described in [Mic08c] and [Mic08b]. A relevant excerpt of the class discription is given with 1.

The class *Policy* has the in Figure 2 described methods to access policies. (Again, the table lists an excerpt of the list given at [Mic08b] restricted to the relevant entries for transferring lagacy policies to an ORKA-based system. The complete list can be found at the cited reference.)

Methods	Description
Condition getCondition(java.lang.String condition)	Gets the condition object identified by name.
java.util.Set getConditionNames()	Get the set of condition names associated with the policy.
java.lang.String getDescription()	Gets the description for the policy.
java.lang.String getOrganizationName()	Gets the organization name under which the policy is created This would be set only for policies that have been read from data store.
Referral getReferral(java.lang.String referralName)	Gets the Referral object identified by name.
java.util.Set getReferralNames()	Get the set of referral names associated with the policy.
Rule getRule(java.lang.String ruleName)	Gets the rule object identified by name.
java.util.Set getRuleNames()	Gets the set of rule names associated with the policy.
Subject getSubject(java.lang.String subjectName)	Gets the Subject object identified by name.
java.util.Set getSubjectNames()	Get the set of subject names associated with the policy.
boolean isActive()	Checks whether the policy is active or inactive An inactive policy is not used to make policy evaluations.
boolean isRealmSubject(java.lang.String subjectName)	Checks if the subjectName is a reference to a Subject defined at the realm.
boolean isReferralPolicy()	Checks whether the policy is a referral policy.
boolean isSubjectExclusive(java.lang.String subjectName)	Checks if the subject is exclusive.
java.lang.String toString()	Gets string representation of the policy object.
java.lang.String toXML()	Gets the serialized policy in XML.

Table 2: Accessing Policy Information

## 2.3 Active Directory

Active Directory is a hierarchical collection of objects that describe network resources. Objects are loosely categorised as being services (often software-based, functional resources), users (representing users of the network and groupings), and generic resources (essentially hardware-based network resources, such as printers). Each object can be richly attributed via typing defined by schema objects (known as characterisation schemas), allowing for fine-grain location and discovery of services based on search criteria. For instance, a desktop PC's object entry in an Active Directory can be related to a characterisation schema representing hardware capabilities, such as CPU type and free hard drive space. These schemas can affect how the object can be manipulated, such that objects with certain schemas may not be deleted, or others may only be able to be deactivated.

### Organisational Structures

A directory can be viewed from 3 distinct levels: the forest structure encapsulates all the objects within the Active Directory, and contains trees which encapsulate domains. A domain can be considered as a set of resources that share the same set of security policies, where all objects in the same domain are subject to domain-wide security policies, and domain-specific authentication. In this way, when a user object is related to a domain, it gains access to the resources within the domain by default. Each object can hold a collection of other objects, providing the notion of a directory hierarchy through organisational units (OUs). These units can also encapsulate policies in a similar fashion to domains, such that these policies can be applied to all objects within an OU, however are more flexible in their definition in that they aren't directly relatable to network resources. For instance, when an OU is granted access to a resource, all objects contained within the OU are also attributed access. Finally, sites are collections of IP subnets that share reliable connections to each other and thus provide another grouping of objects.

#### 2.3.1 Policy Model

Access control in Active Directory is provided for by Group Policy management, and the explicit trust afforded by the hierarchical domain and trees. Group Policy Objects can define settings for resources and users, such as access rights to a resource for a particular user. GPOs can be defined for all levels of the directory, and are applied in a specific order. First, a GPO local to the object is applied, followed by site and domain policy objects, and lastly GPOs pertaining to organisational units for the object.

GPOs themselves define how a particular resource will function for users, either through basic deny or permit access control, or through fine-grained customisation to specific services. By default, these may include Windows Registry settings to be included or removed, and the software permissions available, for a user when accessing a computer resource.

#### 2.3.2 Model Matching

The mapping between the entities of the Directory Service and the ORKA Entities are depicted in Figure 8. As the GPO explicitly defines the access for users, roles and resources it directly maps to the ORKA Policy entity and its sub-entity Permission. The Resources entity on the

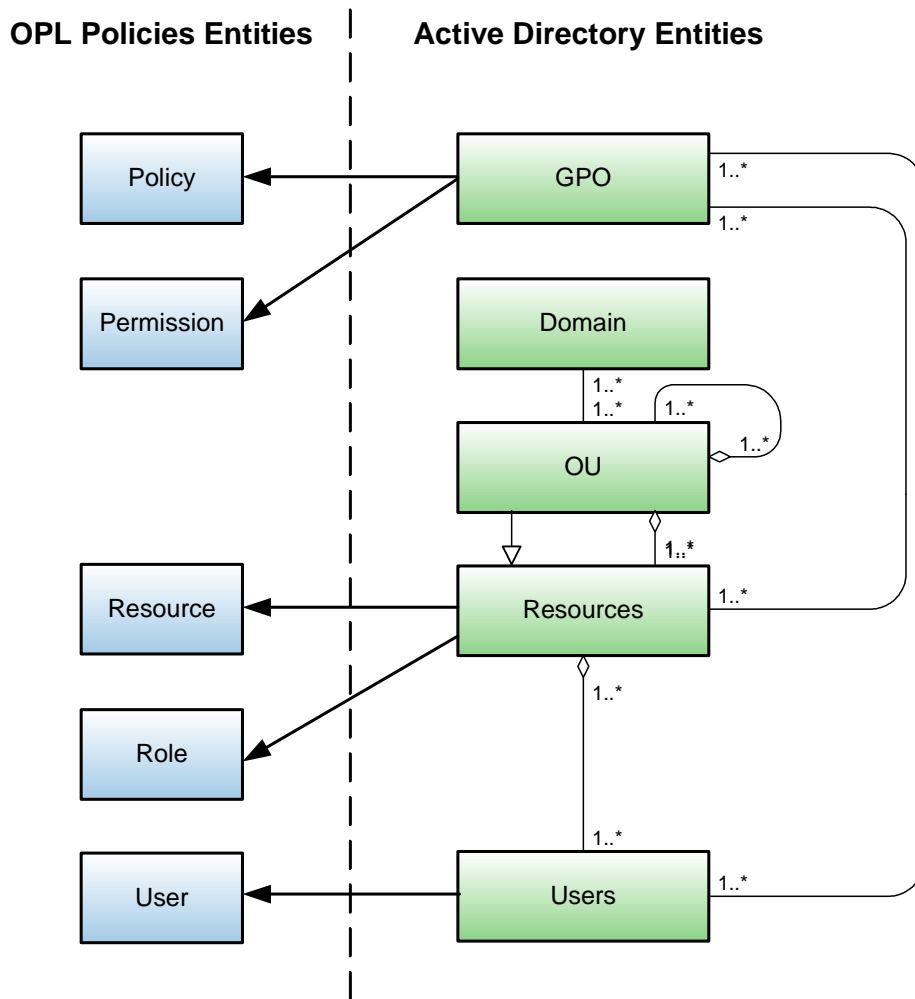


Figure 8: Active Directory Entity Mapping

Active Directory side reflects compared to ORKA both, resources and roles which makes it necessary to map it onto the Resource entity and Role entity and provide some splitting functionality to extract and assign the "Resources" correct to either the Resource entity or Role entity. Users of the Active Directory model map directly to the User entity in ORKA.

### 2.3.3 Interfaces

All editions of Microsoft Windows since Windows NT 5.0 has provided Active Directory services, and as with most Microsoft technology, there are both COM and .NET APIs that allow for the control and manipulation of these Active Directory services. As .NET is intended as a successor to COM, the following will focus on the .NET API for discussion.

Active Directory services are provided for by several systems:

- Windows NT version 5.0, Windows 2000, Windows XP, Windows Vista
- Lightweight Directory Access Protocol (LDAP)
- Novell NetWare Directory Service

- Novell Netware 3.x
- Internet Information Services (IIS)

which are distinguished by path names. For instance, to access an Active Directory residing on an IIS server, one would use the directory prefix IIS:, whereas the operating system Active Directory would use the WinNT: prefix.

Principally, two classes are available for the manipulation and querying of objects in an Active Directory, and are located under the System.DirectoryServices namespace. The DirectoryEntry class encapsulates an object within Active Directory and there exists a characterisation schema for each object, which provides the notion of a classical type for creating and manipulating Active Directory objects. Active Directory objects however, are not directly typed by schema. Instead, attributes can be queried by calling the method invoke on a DirectoryEntry object, and passing the name of the attribute to be returned or set.

```
user.Invoke("Get", new object[] { "Name" });
```

Objects can be located programmatically too, via the DirectorySeacher class, however will only operate over LDAP deployed Active Directories. Filters are used to provide query criteria, and an array is returned with SearchResult objects, relating to the results of the query for attributes (or properties) defined to be returned. For example, the following will search over an LDAP directory, returning all names of users.

```
foo.SearchRoot = new DirectoryEntry("LDAP://dc=sap");
foo.Filter = "(&(objectclass=user)(objectcategory=person))";
foo.SearchScope = SearchScope.Tree;
foo.PropertiesToLoad.Add("Name");
SearchResultCollection bar;
bar = foo.FindAll();
```

```
SearchResult s = new SearchResult();
foreach (r in bar)
{
    Console.WriteLine(objResult.Properties["cn"](0));
}
```

Other classes in the API allow for manipulation of Group Policies, such that they can be created, removed, or objects be assigned to particular domains.

## 2.4 JBossSX - JAAS Security

A key aspect of the Java 2 Platform, Enterprise Edition (J2EE) component models is a simple declarative security model. The EJB and Servlet specifications support a role-based declarative security model that externalizes security from application logic and decouples the application security roles from the deployment environment's security implementation. At the application level, the *ejb-jar.xml* and *web.xml* deployment descriptors define security rules for methods and user roles. The descriptor files *jboss.xml* and *jboss-web.xml* describe the security-domain. Although this model allows for an independent, simple specification of the application server's security requirements, mapping the application-defined security onto the deployment environment

security’s infrastructure is an application-server-specific activity. Thus, configuring a J2EE application’s security requires proprietary application server APIs or tools. One such tool is the Java Authentication and authorisation Service (JAAS).

“Authorisation is the process of verifying that a user has permission to take a specific action. J2EE covers this topic, but it is constrained to role-based authorisation, which means that activity can be constrained based on the roles the user has been given. For example, users in the manager role might be able to delete inventory, while users in the employee role might not.

The EJB and Web containers have a request interceptor architecture that includes a security interceptor, which enforces the container security model. At deployment time, the security domain in the *jboss.xml* and *jboss-web.xml* descriptors is used to obtain the security manager instance associated with the container and used by the security interceptor. When a secured component is requested, the security interceptor delegates security checks to the security manager instance associated with the container. For the JBossSX default security manager implementation, shown in Figure 9, security checks are based on the information associated with the login context [JBo07].

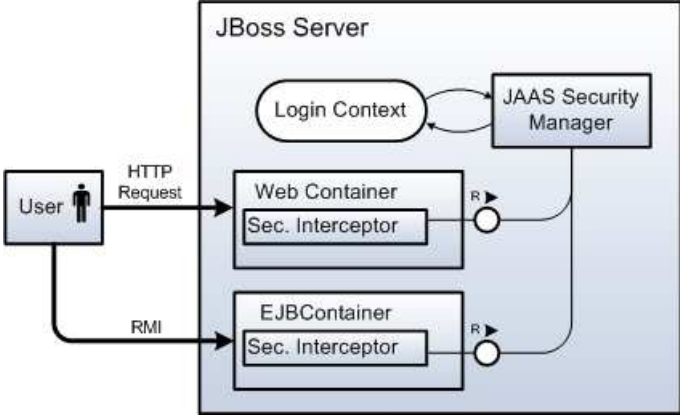


Figure 9: JBossSX Security Interceptor

## Declarative Security

Declarative security refers to the means of expressing an application's security structure, including roles, access control, and authentication requirements in a form external to the application. The deployment descriptor is the primary vehicle for declarative security in web applications.

There are five types of deployment descriptors, each of which corresponds to a type of deployment unit:

- EJB deployment descriptors are defined in the Enterprise JavaBeans specification.
- Web deployment descriptors are defined in the Java Servlet specification.
- Application and application client deployment descriptors are both defined in the J2EE platform specification.
- Resource adapter deployment descriptors for Java Connectors are defined by the J2EE Connector architecture specification.

The Deployer maps the application's logical security requirements to a representation of the security policy that is specific to the runtime environment. At runtime, the servlet container uses the security policy representation to enforce authentication and authorisation. The security model applies to the static content part of the web application and to servlets within the application that are requested by the client. The security model does not apply when a servlet uses the RequestDispatcher to invoke a static resource or servlet using a forward or an include.

## Programmatic Security

Programmatic security is used by security aware applications when declarative security alone is not sufficient to express the security model of the application. Programmatic security consists of the following methods:

- `getRemoteUser`  
The `getRemoteUser` method returns the user name the client used for authentication.
- `isUserInRole`  
The `isUserInRole` method determines if a remote user is in a specified security role.
- `getUserPrincipal`  
The `getUserPrincipal` method determines the principal name of the current user and returns a `java.security.Principal` object.

### 2.4.1 Policy Model

Security enhancements for the Java™ 2 SDK, Standard Edition, v 1.4.2 include the following:

Security constraints are a declarative way of annotating the intended protection of web content. A constraint consists of the following elements:

- Web resource collection  
A web resource collection is a set of URL patterns and HTTP methods that describe a set of resources to be protected. All requests that contain a request path that matches a URL

pattern described in the web resource collection is subject to the constraint. The container matches URL patterns defined in security constraints using the same algorithm described in this specification for matching client requests to servlets and static resources.

- **Authorisation constraint**

An authorisation constraint is a set of security roles at least one of which users must belong for access to resources described by the web resource collection. If the user is not part of an allowed role, the user must be denied access to the resource requiring it. If the authorisation constraint defines no roles, no user is allowed access to the portion of the web application defined by the security constraint.

- **User data constraint**

A user data constraint describes requirements for the transport layer of the client server. The requirement may be for content integrity (preventing data tampering in the communication process) or for confidentiality (preventing reading while in transit). The container must at least use SSL to respond to requests to resources marked integral or confidential. If the original request was over HTTP, the container must redirect the client to the HTTPS port.

### **Example Policy**

```
<security-role>
  <role-name>manager</role-name>
</security-role>

<security-role-ref>
  <role-name>MGR</role-name>
  <!-- role name used in code -->
  <role-link>manager</role-link>
</security-role-ref>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>SalesInfo</web-resource-name>
    <url-pattern>/salesinfo/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
<auth-constraint>
  <role-name>manager</role-name>
</auth-constraint>
<user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

## 2.4.2 Model Matching

The authorisation concepts of JBoss security that are based on JAAS-specification in principle implement a role-based access control model. the mapping is therefore rather straightforward. JBossSX focuses on the protection of web resources. These resources could be directly mapped onto OPL resources. The specified methods are mapped onto OPL permissions. While JBossSX supports communication channel security in terms of data encryption we are only importing authorisation constraints that are related to role memberships and mapped these onto an OPL constraint. The subject list and role assignment definitions are imported from an LDAP directory that is utilized by JBossSX. LDAP groups and nested groups are directly mapped onto an OPL role hierarchy under consideration of the JAAS related security role-link defined in the deployment description files. LDAP users are directly mapped onto OPL users. Attributes are copied as OPL attributes and directly assigned to roles and users. The described mapping is depicted in Figure 10.

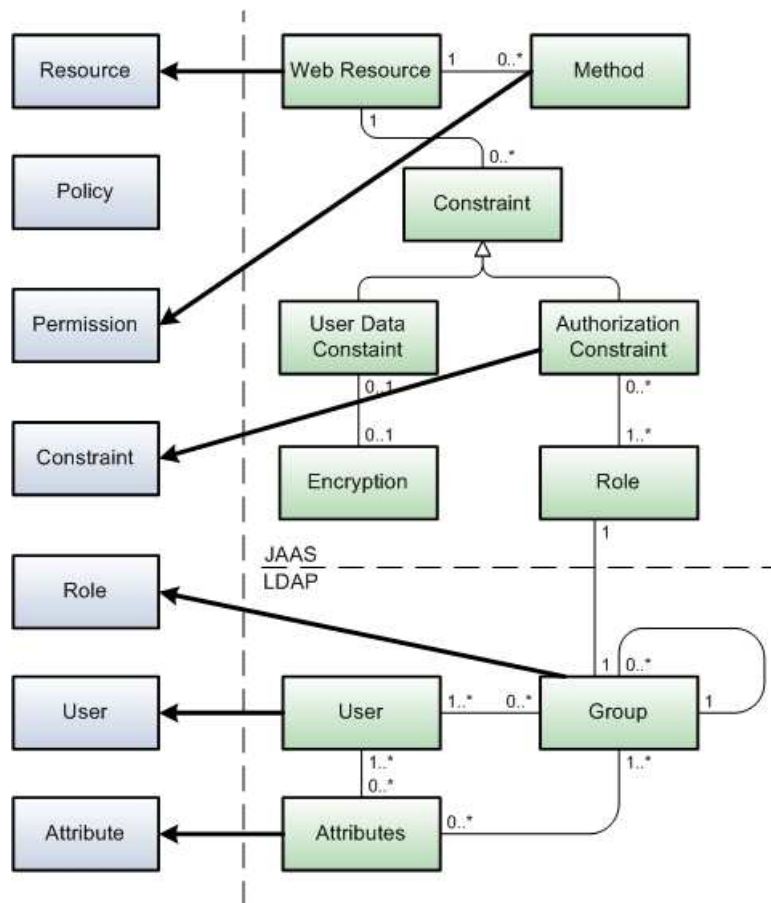


Figure 10: Mapping JBossSX Entities

### 2.4.3 Interfaces

#### JBossSX API

The JBossSX API defines a set of testing and informative method that can be used to query and test security permissions. These methods are contained in the package *org.jboss.security*. While there is at least to our knowledge no tool available for JBossSX to extract descriptive security configurations the following API could be utilized to write a small application that is able to search through the deployment descriptors and stores them in OPL format. The most suitable methods are described in Table 3.

Class	Method
Class AuthorisationInfo	public <i>PermissionCollection</i> getPermissions( <i>Subject</i> subject, <i>CodeSource</i> codesource)
Class AuthorisationInfo	public void grant( <i>CodeSource</i> cs, <i>ArrayList</i> permissions, <i>Principal</i> [] principals)
Class AuthenticationInfo	public <i>AppConfigurationEntry</i> [] getAppConfigurationEntry()

Table 3: JBossSX API Methods

## LDAP

Because JBossSX provides no tool to directly access an LDAP directory for querying user and group data we have to utilize some of the existing LDAP API provided by various API provider, i.e. the *javax.naming.ldap* and *javax.naming.directory* packages. The number of operation that are supported by the LDAP protocol since version 2 is listed in table 4. This list is extend by LDAP v3 by *extended* operations.

Operation	Description
bind	Used to start a connection with the LDAP server. The LDAP is a connection-oriented protocol. The client specifies the protocol version and the client authentication information.
unbind	Used to terminate the connection with the LDAP server.
search	Used to search the directory. The client specifies the starting point (base object) of the search, the search scope (either the object only, its children, or the subtree rooted at the object), and a search filter (RFC 1960). It can also supply other information to control the search, such as the names of the attributes to return and the size and time limits. The search results consist of LDAP entries (and the attributes requested) that satisfy the filter.
modify	Used to modify an existing entry. The client specifies the name of the entry to be modified and a list of modifications. Each modification consists of an attribute and information regarding whether its values are to be added, deleted, or replaced.
add	Used to add a new entry. The client specifies the name of the new entry and a set of attributes for the new entry.
delete	Used to remove an existing entry. The client specifies the name of the entry to remove.
modify	RDN Used to change the RDN of the last component of an existing entry (that is, to assign the entry a new name in the same context). The client specifies the DN for the entry and the new RDN.
compare	Used to test whether an entry has an attribute/value pair. The client specifies the name of the entry and the
name	and value to check.
abandon	Used to terminate an outstanding request.

Table 4: LDAP Operation List

## 2.5 Oracle Access Manager

Oracle Access Manager provides a full range of identity administration and security functions, that include workflow functionality, auditing and access reporting, policy management, dynamic group management, and delegated administration. Oracle Access Manager offers a DMZ-type three-tier architecture to provide a highly secure deployment and runtime protection of data and applications. Of special interest is the *Access System* component, responsible for access control. The Access System supports a variety of access policies, and is fully integrated with an identity management system so that changes in user profiles are instantly reflected in the Access System's policy enforcement.

Access System stores information about configuration settings and security policies that control access to resources in a directory server that uses Oracle-specific object classes. Administrators can use the Access System to protect Web resources and enterprise resources such as J2EE applications, servlets, EJBs, and legacy applications, so it is the Policy Decision Point or PDP (cf. Figure 11). The Access Server provides dynamic policy evaluation as users access resources.

The Access Manager provides Interfaces for LDAP directories or utilizes its proprietary Oracle virtual directory. Optional it can be further configured to support SNMP and NMS to monitor the status of network activities and components. Access Manager authorisations are governed by a policy domain that includes an authorisation expression among a set of default rules that specify how resources for this domain are protected. Administrators work with the Policy Manager system - to define policies that restrict access to specific resources by user, role, group membership (static, nested or dynamic), time of the day, day of the week and IP address.

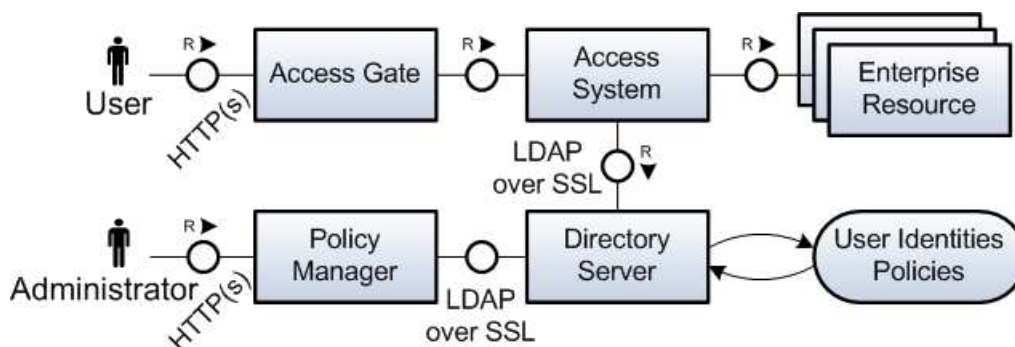


Figure 11: Access System Communication

### 2.5.1 Policy Model

By default, The Access system provides centralized policy-based authorisation services to secure access to web and J2EE resources. Authorisation is governed by a policy domain that includes an authorisation expression among a set of default rules that specify how resources for this domain are protected [Ora06].

In addition, the Access System provides an authorisation API can be used to build custom authorisation plug-ins to allow incorporating custom authorisation logic into the access management policies, which can extend the available range of authorisation options available out-of-the-box. In many cases, the authorisation plug-ins are leveraged to incorporate existing authorisation

logic or systems that customers want to either continue using or migrate from while deploying the Access System.

The Access Manager can be enabled to support J2EE security for hosted applications, and defines URL-policies for enforcing access control over web resources hosted in web deployment container. Using Access Manager based URL-policies for web resources, an administrator specifies such complex policies for the application resources, which are evaluated by the agent in order to ensure that access to these resources is granted only when all conditions are satisfied. An administrator can set policies that govern access to these resources at any level of granularity, such as that for a single user or for an entire organization.

- URL Prefix

The URL prefix is the starting point for resources in a policy domain. A URL prefix defines the beginning boundary of a policy domain, that is, its first resource. A URL prefix maps to a directory on the file system of an application servers or Web servers. All resources under the URL prefix are protected by the default rules of the policy domain unless more specific rules are applied to them through policies. It is possible to assign one or more URL prefixes to a policy domain, but each URL prefix can belong to one policy domain only.

- URL Pattern

URLs for policies specify the fine-grained portion of a resource's namespace. To fully identify the URL, the host identifier and URL prefix for the policy domain are concatenated with the policy's URL pattern. An example of a URL pattern covering many resources is a URL for all HTML pages, such as *\*.html*.

- Authentication Rule

Specifies the method used to challenge and authenticate users requesting access to protected resources. Can specify actions to be taken if authentication is successful or if it fails. Only one default authentication rule can be included in a policy domain. Each of its policies, however, can have its own authentication rule.

- Authorisation Expression

Authorisation expressions include authorisation rules and the operators used to combine them. Combining authorisation rules within expressions to create from simple to complex means of specifying who is allowed or denied access to the protected resources is possible.

- Authorisation Rule

Allows or denies a user access to requested resources within a policy domain or policy. Can specify actions to be taken if authentication is successful or if it fails. These rules are included in authorisation expressions. If more than one rule is included in an expression, the order of evaluation of the rules is determined by the logic specified to form the expression by combining them via *And* or *Or* operations. An authorisation rule can also specify one or more **conditions** for access. Only one default authentication rule can be included in a policy domain. Each of its policies, however, can have its own authentication rule.

- Audit Rule

Rules contain schemes that define how the resources of a policy domain are to be protected, including:

- How authentication of the user is to be performed.

- Whether a user has the right to access a domain resource and any conditions defining access rights. Authorisation rules are included in authorisation expressions. A policy domain must contain exactly one authorisation expression.
- Events to be audited pertaining to the policy domain or policy. Rules can include actions to be executed depending on the result of the evaluation of user information against the specifications of the rule.
- Action  
An Action allows to define an obligation that must be performed by the Access Manager in case a rule or expression evaluates to true or false. For instance URL redirection, accessing cookie data, or hand over information to other applications (i.e. Personalizing the end-user's interaction with the receiving application).

## 2.5.2 Model Matching

Relating to Figure 12 OPL Resources can be derived from the policy domain and patterns that encode URLs to protected entities. OPL Permissions that are part of an OPL Policy are derived from the authorisation Expressions and their related rules. Conditions relate to the similar concepts as OPL constraints in terms of temporal restriction of authorisations. Concrete user information and their related group and role associations can be directly derived from the LDAPA-like directory server.

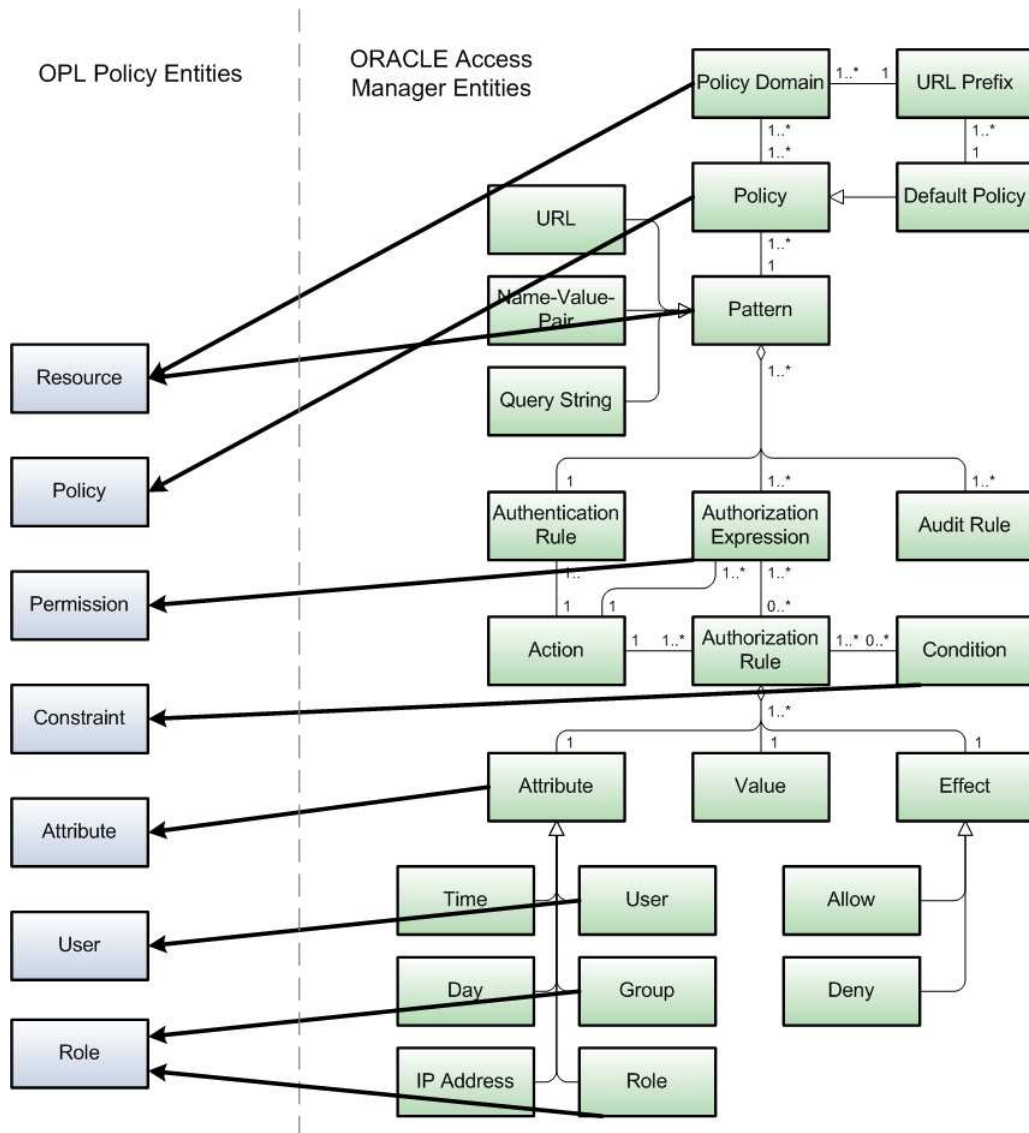


Figure 12: Entity Mapping

### 2.5.3 Interfaces

Oracle Access Manager provides IdentityXML and Policy Manager API that enable writing custom applications and plug-ins to perform Identity System functions and to be able to perform other Access System functions programmatically [Ora07].

#### **IdentityXML**

IdentityXML provides a programmatic interface for carrying out the actions that a user can perform when accessing an Identity System application from a browser. IdentityXML enables external applications to access these Identity System functions:

- **User:** Create, delete, and manage user data within or outside of a workflow or an asynchronous workflow.
- **Group:** Create, delete, and manage groups and subscriptions.
- **Organization:** Create, delete, and manage organization object data.
- **search:** One basic component of many IdentityXML functions is the search operation.

#### **Policy Manager API**

The Policy Manager API provides an interface which enables custom applications to access the authentication, authorisation, and auditing services of the Access Server to create and modify Access System policy domains and their contents. The Policy Manager API provides Java, C, and managed code bindings for classes to instantiate objects such as:

#### **Access Policy Objects:**

These objects are used for data that is part of policy domains, policies, access conditions, audit rules and other policy domain content that Access Administrators ordinarily configure through the Policy Manager.

#### **Access Server Connections:**

The ObAccessManager class represents one or more connections to Access Servers hosting the Policy Manager Service. An application uses ObAccessManager methods to send requests to get and set policy objects and to get configuration objects.

## 2.6 IBM Tivoli Access Manager

IBM Tivoli is a product family from IBM to organize and manage a company's IT-landscape. The operational management comprises products in the fields of asset management, business application management, security management, storage management.

One product is the IBM Tivoli Access Manager which provides policy-based access control for IT-system infrastructures. It builds up on security standards (such as ISO17799 [Ris02]) and the general security architecture model defined by Common Criteria [Cri08] representing "best practices" for security design.

The Tivoli Access Manager has three types of core components:

### 1. *Base components*

These components are generally needed to manage and administrate the environment where the Access Manager is implemented. The components are: A user registry based on a database containing user and group information, an authorisation database which contains the security policies specified for the environment, as well as a Policy Server, a Policy Proxy Server and the general authorisation service which together form the evaluation engine for access requests. Additionally, the base components also include a command-line administration utility and the accompanying administration API as well as a Web Portal Manager to provide browser-based access to administrative operations.

### 2. *Resource managers*

The resource managers are comparable to the Policy Enforcement Points defined within the ORKA project. They are responsible to support the authorisation engine by enforcing the evaluation results given by the engine.

### 3. *Interface components*

The interface components of the Access Manager provide the API to connect to the access manager functions and allows direct programmatic access to the components. The APIs provided are namely: aznAPI ("standard programming and management model for integrating authorisation requests and decisions with applications" [IBM06]), Java API for the integration of Access Manager authorisation functionality into an Java 2 environment, Microsoft .Net interface for authorisations in an .Net-environment (Managed C++, C#, Visual Basic .NET), as well as a management API for C and Java to accomplish administrative interactions. Also possible is to include external evaluation mechanisms for policy evaluation.

### 2.6.1 Policy Model

The policy model comprises the users and groups stored in the already mentioned user repository, the so called protected object space, and three different types of access control policy representations composing the authorisation model. We will discuss each of them in the following paragraphs.

#### **User Repository**

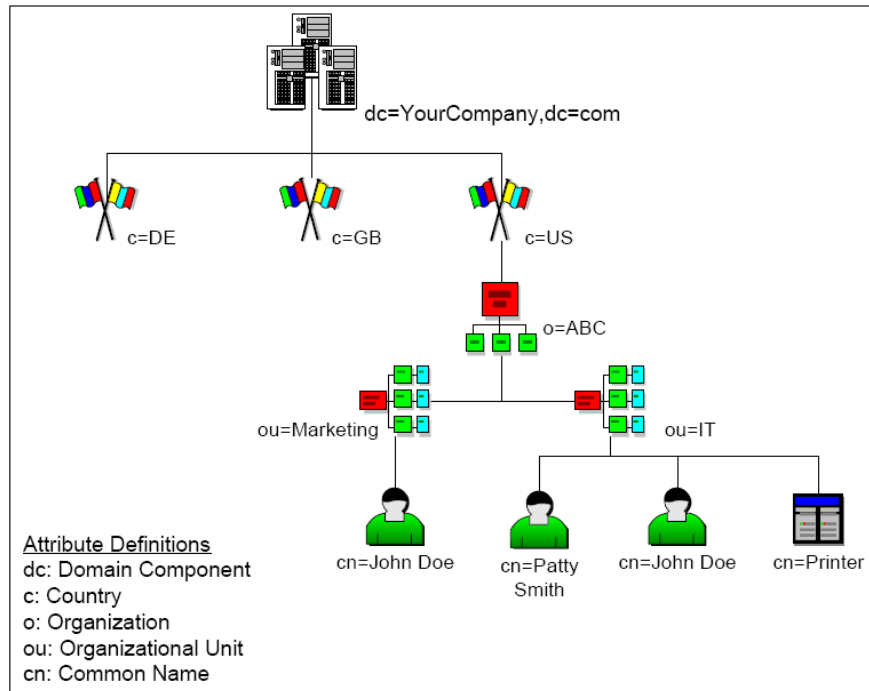


Figure 13: Example of hierarchical structure in an directory service [IBM06]

For the authorisation functionality a user and role repository is required. Tivoli Access Manager uses a database of user identities, roles (called 'groups'), and additionally stored metadata. The idea behind the user repository is not necessarily the authentication of a user, but map authenticated users (authenticated via various possible mechanisms such as certificate-based authentication) onto the identities defined in the repository for authorisation operations.

The registry structure in a default installation compares to an LDAP-based directory. The following image (taken from [IBM06]) illustrates the Tivoli directory model on a simple example.

### Authorisation Model

The authorisation data is separated from the user and role registry, stored in a special database: "Called the protected object space, it uses a proprietary format and contains object definitions that may represent logical or actual physical resources." [IBM06] In this table different types of security mechanisms are stored to specify the policies for protecting objects and other resources. The security mechanisms to define such policies are a combination of three different categories:

1. *Access control list (ACL) policy templates*

An access control list specifies which identities (= users in most cases) and groups (stored in the user repository) can be considered for a access on an object using a predefined action. An ACL is attached to an object in the Protected Object Space (see below). It explicitly names the user or group and the specific operation for the object it is attached to.

2. *Protected object policy (POP) templates*

A Protected object policy specifies conditions for the protected objects to access them. The conditions relate to privacy, integrity, auditing, and time-of-day access. The protected

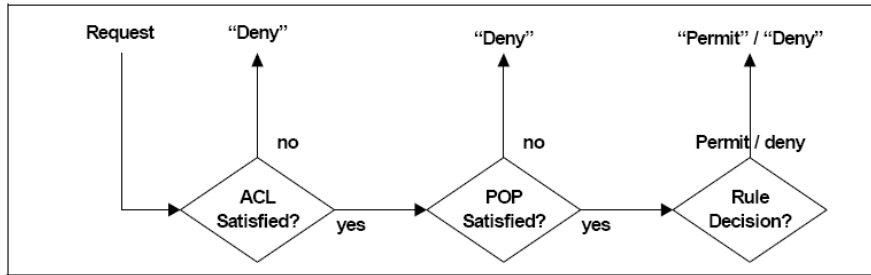


Figure 14: Authorisation decision flow [IBM06]

object policy explicitly does not specify any action or user (for which access should be granted or denied) on the resource but defines more general conditions on the context of the object which have to be fulfilled that access is granted.

Conditions which can be applied are given in [IBM06], stating that certain POP attributes may be required such as audit level, or time-of-day.

### 3. Authorisation rules

With authorisation rules complex conditions for access decisions for protected objects can be specified which have to be met before access is granted. Context data used for such decisions can be based on the request context, the current environment, or external factors.

For the evaluation of rules an XSL parser is used as rules evaluation engine. The input for the engine are two files: An **XML document** and **XSL document**. The XML document contains the so called authorisation decision information (ADI)<sup>3</sup>. The XSL document contains the configured rule for the accessed object.<sup>4</sup>

Whenever the XSL parser gets an XML document as input the XSL parser formats the XML document using the XSL document. As with the parsing of the XML file an access decision should be made, the rules (withing the XSL document) must be written such that the output of the parsing process will be TRUE in case access should be permitted, FALSE in case access should be DENIED, or INDIFFERENT if no decision can be made. In other words, the result of the XSL parser (respectively the rule evaluation engine) is the result of the formatting of the inputted XML document. [IBM06]

There is a general authorisation process which defines in case of an access request, in which order the three types of access control policies (Access Control Lists, Policy Object Policies, and authorisation Rules) are evaluated. If during this process the evaluation of the policies of one policy type returns that permit should be denied, the process is interrupted and the DENIED-answer is given back. The authorisation flow is depicted in 14 below.

<sup>3</sup>In ORKA the equivalent would be an access request context.

<sup>4</sup>In ORKA the equivalent would be the policy definition.

## Protected Object Space

The IBM Tivoli Access Manager architecture represents objects to be protected in an Protected Object Space. It is a hierarchical, logical representation of objects divided into two types:

- Resource Objects, and
- Container Objects.

Resource Objects are a logical representation of physical objects to be protected. This can be files, services, Web resources, message queues, etc. Container objects are a logical construct to represent groups of resource objects or groups of other container objects. This allows to specify a hierarchical structure among the resource objects. The root node for the complete Protected Object Space is always "/".

It is possible to implement the IBM Tivoli Access Manager in a multi-domain landscape. In this case there will be several authorisation repositories and resource managers; (at least) one of them for each domain. Hence, every Container Object (including the root node) and every Resource Object definition is part of a security domain.

Each of the elements in the object space can be referenced by security policy specifications (i.e., either the ACL, the POPs, or the authorisation rule specifications. Related to ORKA and the respective mapping from Tivoli Access Manager policy entities to ORKA policy entities, this means the resource entity defined within an ORKA policy may result from either a resource object directly or a container object respectively.

### 2.6.2 Model Matching

In Figure 15 the mapping between the the Tivoli Access Manager Entities and the OPL Policy Entities is shown.

As described above, for the Tivoli Access Manager the Protected Object Space comprises Container Objects and Resource Objects. They map to the Resource Entity on the ORKA side. The authorisation rules map on to the general policy entity and gives information for the attribute related entity. The ACL list is a list of permissions of actions on resources and therefore maps to the Permission Entity which is part of the general ORKA Policy. For users and groups the mapping is to the users and roles on the ORKA side respectively.

### 2.6.3 Interfaces

IBM Tivoli Access Manager provides an Java application interface for administration of users and groups as well as for the administration of access control specification. In this section we will show which functionality of the interfaces provided can be used for accessing the information stored within the Access Manager. For accessing user and policy information of a running system, a Tivoli Access Manger user has to be available which has the appropriate rights to access the data. The descriptions given next are taken from [?], though we only describe the functions relevant for the legacy import for ORKA.

For the administrative access to user and group data, the Java classes **PDUser** and **PDGroup** are available having the following functions:

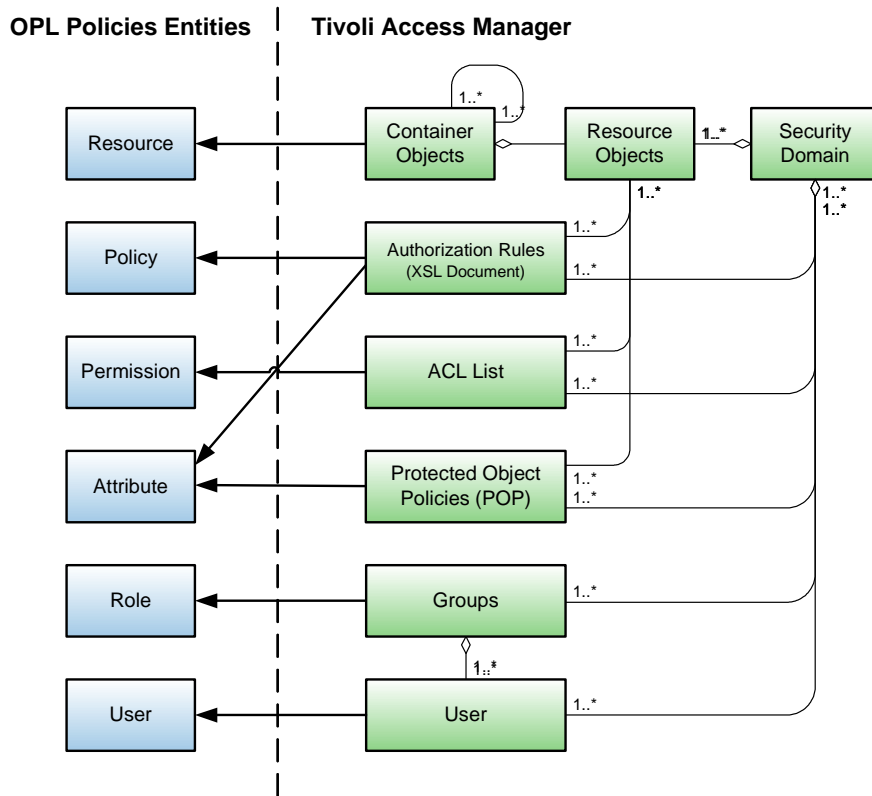


Figure 15: Tivoli Access Manager Entity Mapping

Methods	Description
PDUser constructor	Instantiates a user object for the specified Tivoli Access Manager or user registry
PDUser getDescription	Returns the user description.
PDUser getRgyName	Returns the user registry name for the user.
PDUser getFirstName	Returns the first-name attribute for the user.
PDUser getLastName	Returns the last-name attribute for the user.
PDUser getPolicy	Returns the password and account policy settings associated with the user.
PDUser getGroups	Lists the groups in which the user is a member.
PDUser isAccountValid	Returns the account-valid indicator for the user.
PDUser isPDUser	Returns an indicator whether this is a Tivoli Access Manager user.
PDUser.listUsers	Lists Tivoli Access Manager users.
PDGroup constructor	Instantiates a group object for the specified Tivoli Access Manager or user registry name.
PDGroup getDescription	Returns the group description.
PDGroup getMembers	Lists the members of a group.
PDGroup.listGroups	Lists Tivoli Access Manager groups.

Table 5: Accessing User and Group Information

Methods	Description
PDProtObject getAclId	Gets the name of the ACL attached to the specified protected object.
PDProtObject getEffectiveAclId	Gets the name of the ACL in effect for the specified protected object.
PDProtObject getPopId	Gets the name of the POP attached to the specified protected object.
PDProtObject getEffectivePopId	Gets the name of the POP in effect for the specified protected object.
PDProtObj getAuthzRuleId	Gets the name of the authorisation rule object that is attached to the specified protected object.
PDProtObj getEffectiveAuthzRuleId	Gets the name of the authorisation rule object that is in effect for the specified protected object.
PDProtObject getDescription	Gets the description of the specified protected object.
PDProtObj.listProtObjects ByAuthzRule	Lists the protected objects that have the specified authorisation rule attached.

Table 6: Accessing Object Information

For the administrative access to protected object definitions, the Java class **PDProtObject** is available and has the following functions:

For the administrative access to ACL, POP and authorisation rules definitions, the Java classes **PDACL**, **PDPOP**, and **PDAuthzRule** are available having the following functions:

Methods	Description
PDAcl constructor	Instantiates the specified ACL.
PDAcl getDescription	Returns the description of the specified ACL.
PDAcl getId	Returns the name of the specified ACL.
PDAcl.listAcls	Returns the names of all the defined ACLs.
PDPop getDescription	Returns the description of the specified POP.
PDPop getId	Returns the name of the specified POP.
PDProtlistProtObjectsByPop	Finds and lists all protected objects that have the specified POP attached.
PDPop constructor	
PDProtObject getPop	Returns the specified POP object.
PDPop.listPops	Lists all POP objects.
PDPop getIPAuthInfo	Returns the IP authentication level information from the specified POP.
PDPop getAuditLevel	Returns the audit level for the specified POP.
PDPop getQOP	Returns the quality of protection (QOP) level for the specified POP.
PDPop getTodAccessInfo	Returns the time of day range for the specified POP.
PDPop getWarningMode	Returns the warning mode value from the specified POP.
PDPop getAttributeValues	Gets the values for the specified extended attribute from the specified POP.
PDPop getAttributeNames	Lists the extended attributes associated with the specified POP.
PDAuthzRule constructor	Instantiates the specified authorisation rule object.
PDAuthzRule getId	Returns the ID for the specified authorisation rule.
PDAuthzRule getDescription	Returns the description for the specified authorisation rule.
PDAuthzRule.listAuthzRules	Lists all of the registered authorisation rules.

Table 7: Accessing Policy Information

## 3 Legacy Adapter Tool

As initially described in the introduction it is unlikely that it is feasible to replace an existing security infrastructure by the ORKA architecture without considering any existing security configuration, i.e. user data directories, that might be in place.

Recent analysis in work package 3.1 and the analysis that was done in the course of this work package reveals, that there are various kinds of legacy systems available in a modern enterprise system landscape, providing different methodologies and interfaces to persist, access, and query security configuration. An interesting observation is that in general all systems support to some degree the role-based access control model. Another observation is the huge support of LDAP or LDAP-like directory services. Some applications even support JASS security concepts to provide fine-grained method-based access control for Enterprise Java Beans.

Based on these two observations we propose a modular security configuration adapter tool that is able to connect to a legacy security infrastructure, query existing user and authorisation data, maps the underlying data model to the OPL data model, and translates the legacy data sets into OPL data sets that can be directly used within the ORKA administration tool. As a second option the legacy adapter can be run in *offline* mode. This mode does not generate OPL entities for the administration tool, but rather writes OPL entities as OPL/XML files. In this way, OPL/XML files can be read and translated into OPL for testing purpose without an existing legacy system or in case the legacy system is not available any more.

### 3.1 Design Concepts

The ORKA legacy adapter is designed to support a various set of different legacy systems. Therefore, each adapter is considered an independent component, consisting of two sub components. The *Connector* module is a hand tailored component to connect to exactly one type of legacy system. The *Translator* component is used align a legacy data set with an OPL data set or entity. The legacy adapter is configured via a simple configuration property file. The overall idea is to keep the adapter as independent from the other ORKA administration components as possible in order to enable an early start of the adapter development. Later, when the ORKA administration components are complete and the OPL API is defined, the legacy adapters can be integrated into the administration tool itself.

Therefore, the design is based on the following key concepts:

- **Modularity**

In order to support various kinds of legacy system integration scenarios we follow a modularised approach. Therefore, we define a set of abstract classes that must implement a connection and a translation interface. These interfaces are designed to support role-based access control authorisation querying and the *OPL RBAC Core* Module. Thus, it is possible to query user data, their role assignments, and related permission assignments.

According to *OPL RBAC Core* the corresponding OPL entities can be generated by the *Translator*. The *Translator* maps imported elements of the legacy system policy data structure to OPL elements according to the model matching of the previous sections.

- **Offline Mode**

The offline mode concept allows to dump legacy security policy configurations into a file and import security policies stored in OPL/XML into the ORKA administration tool using the *Translator* component. This concept fosters the idea of importing legacy policy configurations beforehand, even without a running ORKA infrastructure. This allows to shut down legacy systems early in case they must be replaced or are likely to fail. Another benefit is the unit testing capabilities of the *Translator* without a legacy system in place. This allows to verify the OPL generation for the ORKA administration tool independently from any legacy system available.

- **Usage of OPL API**

The *Translator* components are designed to generate OPL or OPL/XML entities and to write them into the ORKA administration tool system memory or into a file in case of offline mode. Instead of reimplementing OPL creation functionality that is also provided by the ORKA administration tool via an OPL API, the *Translator* creates OPL policies by using the administration OPL API provided by the ORKA administration tool. Therefore, the *Translator* can be considered as a batch processor for OPL policy generation by specifying the output format and automatically calling OPL API methods. Accordingly, a conceptual schema is shown in Figure 16.

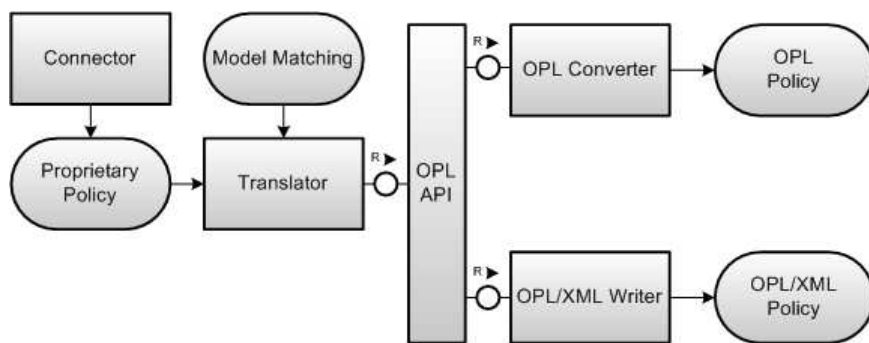


Figure 16: Policy Translation

### 3.2 Integration Scenario

Conceptually, the legacy adapter components are part of the ORKA administration tool. As shown in Figure 17 an administrator works directly with the ORKA administration tool in order to specify OPL-based policies. During administration OPL policies are stored in an intermediate format in the system memory of the administration application for faster policy editing. When a policy is stored and ultimately transferred to the policy storage service (PSS) the *OPL Converter* transforms the memory-based representation of an OPL policy into the persistence format OPL/SQL. Analogously the *Converter* transforms an OPL policy into the validation format OPL/Isabelle or OPL/XACML based on the 1-Step-Transformation concept discussed in work package 1.3.

The legacy import modules are part of the ORKA administration tool and can be used either directly via the administration UI or stand alone (availability of the OPL API is still mandatory). Each import module consists of two sub components. A connector and a translator. The connector component is responsible for establishing a connection to the legacy application. Usually,

the connection will be based on RMI, SOAP or other XML-based message exchange. Necessary connection information, such as service endpoint URI, user or password data is stored in a plain text configuration file.

The *Translator* component is used to query policies from the legacy application via the dedicated *Connector*. Based on an internal mapping description the retrieved legacy policies are translated by calling the related OPL creation methods of the OPL API. Therefore, the necessary method parameters are extracted from the imported legacy policies. Due to investigation of various legacy system we restrict this mapping and translation capabilities to *OPL RBAC Core* policies. In case the import module is called from the administration tool UI, the imported policies are directly loaded into the administration tool.

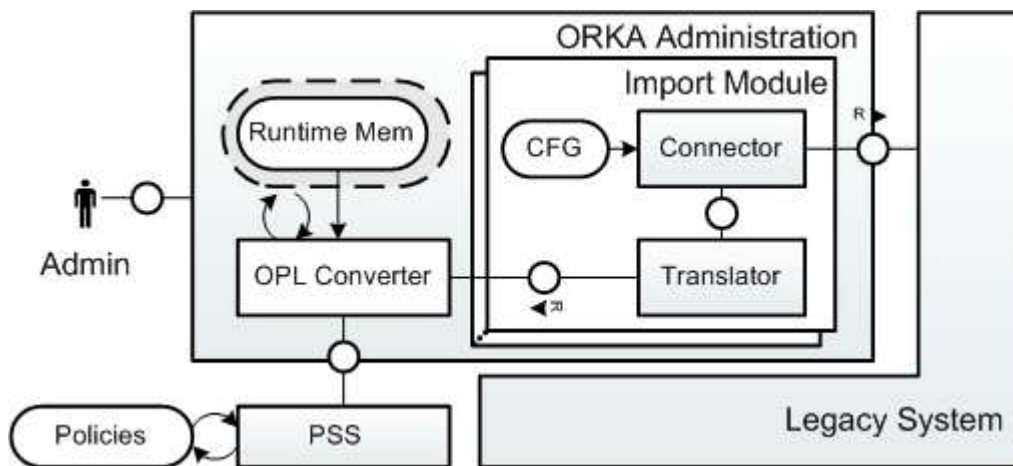


Figure 17: Administration Integration

### 3.3 API Definition

In this section we outline the basic class concepts of the legacy adapter tools. As mentioned earlier each adapter consists of a *Connector* and *Translator* component. Because each legacy system is likely to provide its own interfaces to remote access, the *Connector* component must be rewritten for each accessed legacy system. To provide a general interoperability between the administration tool and the import modules each *Connector* must implement the *Connector Interface*. Similar, the *Translator* must be rewritten for each legacy application in order to extract the necessary information from the imported legacy policies based on the *Translator Interface*. OPL API methods are called in order to create *OPL RBAC Core* policy fragments. The legacy adapter provides a small set of methods that can be called from the administration tool. A short overview of the related class concept is shown in Figure 18. In what follows we give a short list of available methods:

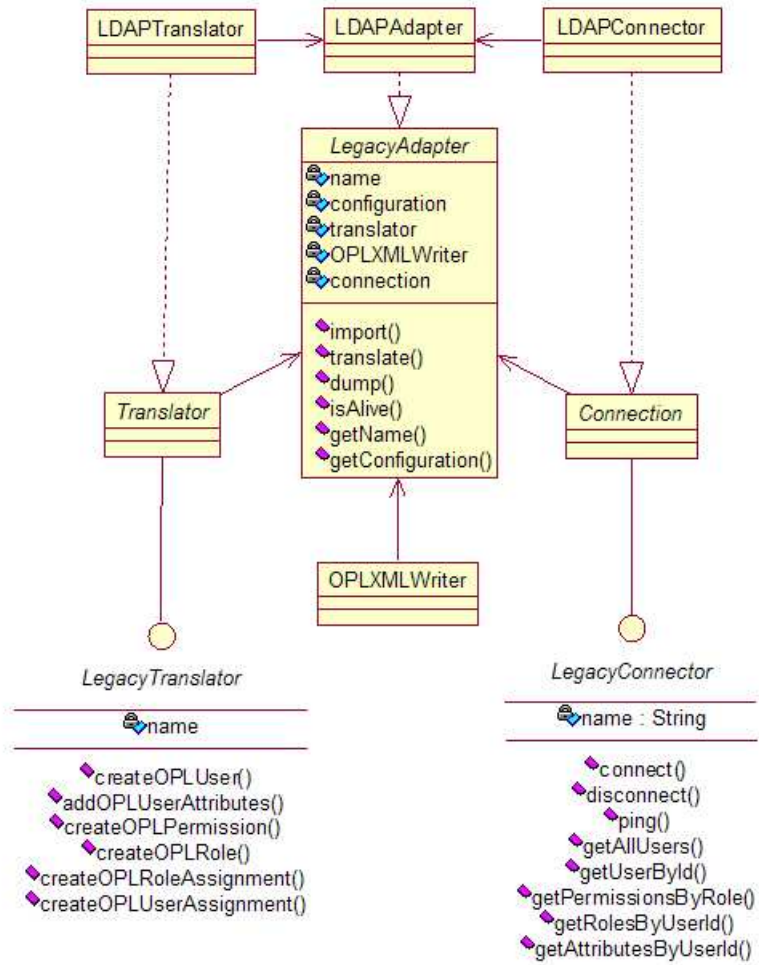


Figure 18: LegacyAdapter Class Concepts

- LegacyAdapter

*Members*

- String **name**: Name of the adapter
- String[] **configuration**: Configuration properties
- Translator **translator**: Reference to Translation entity
- Connection **connection**: Reference Connection entity
- OPLXMLWriter **writer**: Reference to OPLXMLWriter entity

*Methods*

- Document textbfimport():  
Reference to OPLXMLWriter entity
- translate(Document xml,XSLTInputSource xslt)**:  
transforms legacy policies into OPL policies
- dump(String filename, Document xml)**:  
writes OPL/XML policies
- boolean **isAlive()**:  
Returns true if connection is still alive
- String **getName()**:  
Returns the adapter name
- HashMap **getConfiguration()**:  
Returns a configuration hash

- LegacyTranslator

*Members*

- String **name**: Name of the adapter

*Methods*<sup>5</sup>

- Document **createOPLUser(String id, String surname, String firstname)**:  
creates an OPLUser
- Document **createOPLUserAttributes(String id, HashMap map)**:  
creates a user attribute
- Document **createOPLPermission(String perm\_id, String operation\_id, object\_id)**:  
creates a permission
- Document **createOPLRole(String id, String desc)**:  
creates a role
- Document **createOPLRoleAssignment(String perm\_id, String role\_id)**:  
Assigns permission to role
- Document **createOPLUserAssignment(String user\_id, String role\_id)**:  
Assigns user to role

---

<sup>5</sup>Please note that at the moment the OPL API is not defined. Therefore, we consider each OPL method results in an OPL/XML fragment. This must be changed later.

- LegacyConnector

#### *Members*

-String **name**: Name of the connector

#### *Methods*

-ConnectionCode **connect(HashMap connection)**:

Returns ConnectionCode.Open in case system is responsive

-ConnectionCode **disconnect()**:

Returns ConnectionCode.Closed for successful system sign off

-boolean**ping()**:

Returns true if connection is okay

-String[]**getAllUsers()**:

Returns a list of user IDs

-Document**getUserById(String)**:

Returns true if connection is okay

-String[]**getAllRoles()**:

Returns a list of role IDs

-Document**getPermissionByRole(String)**:

Returns a list of permissions for a role

-String[]**getRolesByUserId(String)**:

Returns the role assignments of a user

-String[]**getAttributesByUserId(String)**:

Returns a list of user attributes

- OPLXMLWriter

#### *Members*

-fileWriter**writer**:

Reference to file writer

-Document**document**:

Reference to an XML document

#### *Methods*

-**write(Document document)**:

Write XML document to disk

-Document **read()**:

Reads XML document from disk

## 4 Conclusion

In the course of this paper we have been investigating various kinds of enterprise-scale legacy application and services, such as Active Directory services, SAP Web Application Server, or IBM Tivoli Access Manager. All these system support role-based authorisation concepts. Some even provide finer-grained mechanisms, such as Access Control Lists or limiting constraints.

For every examined legacy application we analysed their underlying policy model and mapped each model entity to OPL entities. Because all these systems support RBAC this is straight forward. In addition, for each system we provide a set of API or system calls to access and query authorisation policies and user data remotely.

Based on the evaluation and analysis of several legacy applications and their interfaces for remote access we designed a modular legacy adapter concept that can be integrated into the ORKA administration tool to support policy import for legacy systems. Therefore, each adapter comes with a *Connector* using the legacy system API method calls to extract role-based configurations. A second component, the *Translator*, is used to map these legacy policy fragments to ORKA OPL/XML fragments. These fragment can be either written to the file system or can be directly used within the ORKA administration tool. By referring to ORKA OPL API we avoid a redundant implementation of the OPL API.

In the follow up work package 3.10 we will provide some reference implementation of these concepts in order to import active directory, LDAP and Java-based security information into ORKA OPL.

## References

- [Cri08] Common Criteria. Common criteria. <http://www.niap-ccevs.org/cc-scheme/>, 2008.
- [IBM06] IBM. Enterprise Security Architecture Using IBM Tivoli Security Solutions, 2006.
- [JBo07] JBoss. Jboss security framework. <http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossSX>, 2007.
- [Mic08a] Sun Microsystems. Sun java system access manager 7.1 administration guide. <http://docs.sun.com/app/docs/doc/819-4670?q=Access+manager>, 2008.
- [Mic08b] Sun Microsystems. Sun java system access manager 7.1 api. <http://docs.sun.com/source/819-4682/index.html>, 2008.
- [Mic08c] Sun Microsystems. Sun java system access manager 7.1 developer's guide. <http://docs.sun.com/app/docs/doc/819-4675/aduom?a=browse>, 2008.
- [Mic08d] Sun Microsystems. Sun java system access manager 7.1 technical overview. <http://docs.sun.com/app/docs/doc/819-4669>, 2008.
- [Ora06] Oracle. Oracle access manager - introduction. [http://www.oracle.com/technology/products/id\\_mgmt/coreid\\_acc/index.html](http://www.oracle.com/technology/products/id_mgmt/coreid_acc/index.html), 2006.
- [Ora07] Oracle. Oracle access manager - developer guide 10g. [http://www.oracle.com/technology/products/id\\_mgmt/coreid\\_acc/index.html](http://www.oracle.com/technology/products/id_mgmt/coreid_acc/index.html), 2007.
- [Ris02] RiskServer. Window to Security Risk Analysis, ISO 17799, Information Security Policies, Audit & Business Continuity, 2002. <http://www.riskserver.co.uk/>.
- [SAP05] SAP. Sap identity management apis. SAP SDN, 2005.