



Deliverable AP 3.3

# **ENFORCE: Policy Enforcement**

Basic Prototype Implementation

14.4.2008

Ute Gerlach, Parks Informatik GmbH  
Stefan Kowski, Parks Informatik GmbH  
Mathias Kohler, SAP AG

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

© 2006 ORKA Consortium



Federal Ministry  
of Education  
and Research

Internal document information:

\$Id: del-ap3.3.tex 999 2008-03-28 09:06:50Z gerlach \$

# Contents

- 1 Introduction 4**
  
- 2 Platform Architecture 4**
  - 2.1 Security Architecture . . . . . 4
    - 2.1.1 Authentication and Access Control . . . . . 4
    - 2.1.2 Role Based Access Control (RBAC) . . . . . 5
    - 2.1.3 Business Application Integration . . . . . 5
  - 2.2 Platform Environment . . . . . 6
    - 2.2.1 Network Communication Protocols . . . . . 6
    - 2.2.2 Cryptographic Functions . . . . . 7
    - 2.2.3 Logging . . . . . 7
  
- 3 Platform Services 8**
  - 3.1 Policy Enforcement Point (PEP) . . . . . 8
    - 3.1.1 User Authentication . . . . . 8
    - 3.1.2 Application Policy Enforcement . . . . . 9
    - 3.1.3 Workflow Policy Enforcement . . . . . 9
    - 3.1.4 Legacy System Policy Enforcement . . . . . 9
  - 3.2 Policy Decision Point (PDP) . . . . . 9
    - 3.2.1 Policy Object Selection . . . . . 10
    - 3.2.2 Policy Translation . . . . . 10
    - 3.2.3 Policy Validation . . . . . 10
  - 3.3 Policy Storage Service (PSS) . . . . . 10
  - 3.4 Policy Context Service (PCS) . . . . . 10
  
- 4 Programming Guidelines 11**
  - 4.1 Repository Directory Structure . . . . . 11
  - 4.2 Software Development Environment . . . . . 11
  - 4.3 Java Package Structure . . . . . 12
  
- 5 Policy Enforcement Prototype 13**
  - 5.1 XACML . . . . . 13
    - 5.1.1 Request - Response Architecture . . . . . 14
    - 5.1.2 XACML Request . . . . . 14
    - 5.1.3 XACML Response . . . . . 15
    - 5.1.4 XACML Policy Structure . . . . . 16
  - 5.2 XACML Use Case Policy Example . . . . . 19
  - 5.3 Runtime Environment . . . . . 25
  - 5.4 Software Testing . . . . . 26
  
- 6 Conclusion 26**

# 1 Introduction

In this work package an initial prototype of the policy enforcement and decision components on base of the drafts and definitions of work package 3.2 will be implemented.

Main part of this work package is the development of the platform architecture with its security basics of authentication and access control and its platform environment with network communication protocols, cryptography and loggings. Furthermore the detailed description of the platform services namely Policy Enforcement Point (PEP) and Policy Decision Point (PDP) will be elaborated.

A special emphasis on the Policy Enforcement Point is to design the user interfaces, the user authentication and the detailed development of application and workflow systems. At the Policy Decision Point we focus the selection of policies and their conversion within the complete ORKA ensemble.

For implementing the components it is necessary to define programming guidelines and code conventions. Therefore we will describe the structure of the repository and the general software development environment.

Furthermore we will start with the implementation of the prototype. To get an ORKA enforcement language we decided to use the language XACML (eXtensible Access Control Markup Language). First we will give an short introduction about the main elements and specifications of XACML. Additionally we will describe the integration of XACML in the ORKA project.

## 2 Platform Architecture

The ORKA platform architecture specified in the part of the document covers the topics authentication and access control, role based access control and business application integration.

### 2.1 Security Architecture

This chapter describes the security architecture of the ORKA policy enforcement system.

#### 2.1.1 Authentication and Access Control

Each user has to identify himself before he can use ORKA-enabled applications. During logon, the user has to provide a password. After he has been logged on successful, the system will retrieve the user's roles and permissions from the Policy Storage Service (PSS).

The user master data can be defined in the PSS. Additionally, it is possible to use an LDAP directory as user repository. In this case, the user information will be retrieved from the directory.

ORKA provides multiple access control interfaces to be able to support different types of applications. A functional API can be used by applications check the permissions of a user on the machine where the ORKA PDP resides (this API will be used by the ORKA administration).

A web service interface can be used by remote services to perform access control checks.

Each logon operation and all failed access control requests will be logged by the ORKA PDP. Additional operations, e.g. successful access requests, may be logged to. A detailed specification of operations and logging configuration will not be defined within the ORKA prototype system, but can easily be added for production use.

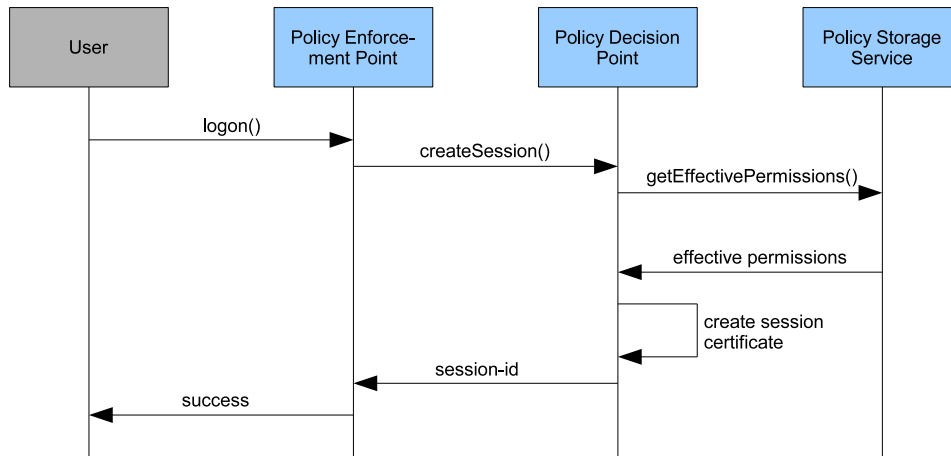


Figure 1: Activity Logon - Authentication - Authorization

### 2.1.2 Role Based Access Control (RBAC)

ORKA uses an role based access control (RBAC) model to control user authorization. Each permission can be assigned to roles, and roles can be assigned to users. Therefore an ORKA user inherits all permissions of the roles that have been assigned to him.

Additional constraints may be specified in the policy to allow finer-grained access control.

The ORKA policies are being specified in the ORKA Policy Language (OPL). OPL is a format policy language that has an XML representation. The XML representation will be used in the enforcement system.

The detailed specification of the ORKA Policy Language can be found in the deliverable of ORKA work package 2.4.

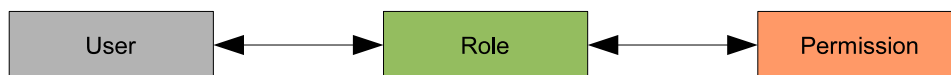


Figure 2: Simplified RBAC model

### 2.1.3 Business Application Integration

ORKA implements different means for integrating business applications, depending on the application type.

Desktop applications performing access checks themselves may call ORKA functions directly, linking to ORKA function libraries. Web applications may use the ORKA web service interfaces to perform access checks.

Workflow systems such as JBPM can be integrated providing callback functions to workflow events like task switch. In the callback functions, ORKA access check functions can be called.

Legacy application, that perform their policy enforcement autonomously, cannot be modified to redirect access checks to ORKA. For this applications, translators can be developed that convert the ORKA system policy to a policy that can be understood by the legacy system. In this case, we assume that some type of policy constraints cannot be translated, therefore we expect an impedance mismatch between the ORKA and legacy system policy.

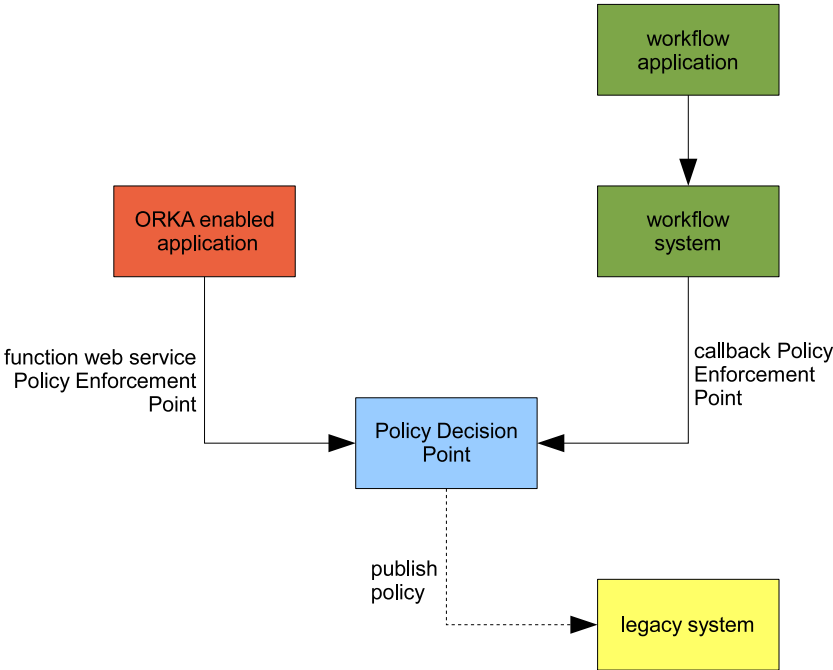


Figure 3: ORKA request with callbacks and external enforcement

## 2.2 Platform Environment

This chapter describes the platform environment of the ORKA policy system. It covers the topic area network communication, cryptographic services and logging.

### 2.2.1 Network Communication Protocols

The ORKA prototype will be implemented as a J2EE application with a web-based frontend. Therefore, all server parts will run on the same machine and only intra-host communication is being used. The ORKA service will call its components by function libraries that do not use network communication. So encrypted communication is not required.

The web frontend will connect to the ORKA server using the HTTP protocol over an encrypted (TLS-) network connection. Using this type of connection, no data on the link can be read or modified by eavesdroppers.

Web applications that use the ORKA web service interface will also use HTTPS connections, therefore no additional encryption is required.

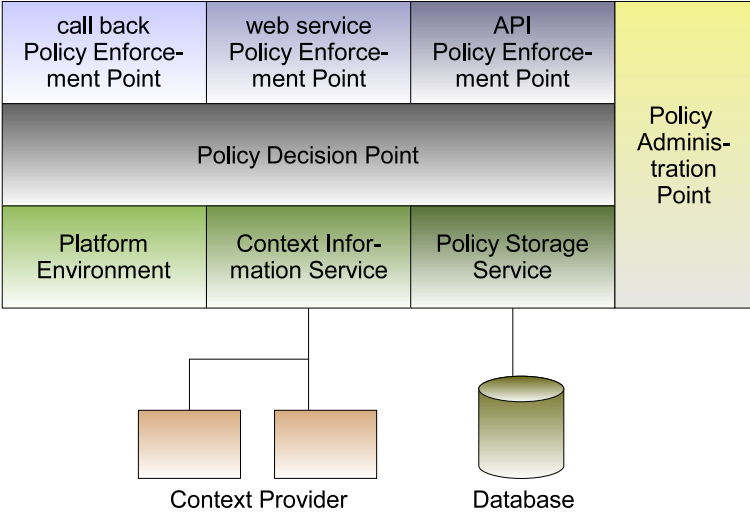


Figure 4: Application architecture

**2.2.2 Cryptographic Functions**

ORKA needs cryptographic and message digest functions to encrypt and store sensitive information. As an example, the password of all ORKA users will be one-way encrypted using a message digest function.

All functions needed are being provided by the standard Java package java.security. If ORKA is to be ported to other programming languages, other packages have to be evaluated and integrated.

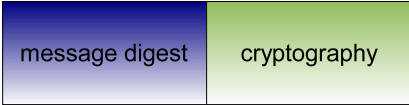


Figure 5: Cryptographical architecture

**2.2.3 Logging**

As ORKA needs to make sure that an auditor finds out about illegal accesses and to maintain consistent processes, all failed access requests and all administrative actions will be logged. Other events may be logged at will.

The system uses the standard java.util.logging package to write messages into text files.

Specialized logging facades will provide easy access to logging functions without exposing the raw log file.

In a production system, all log entries will have to be digitally signed to avoid log forging. The ORKA prototype will not implement log signing.

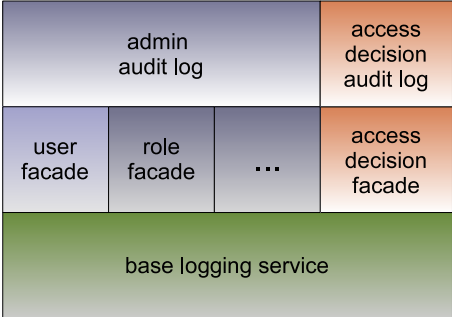


Figure 6: Logging architecture

### 3 Platform Services

This section describes the various enforcement services that will be implemented in the ORKA prototype.

#### 3.1 Policy Enforcement Point (PEP)

The Policy Enforcement Point (PEP) is the link between the business applications and the ORKA access control system. It is responsible for user authentication and provides access check functions for use by ORKA-enabled applications.

In the prototype implementation, we will implement different PEPs: a functional API PEP, a web service PEP and a callback-based PEP for the JBPM workflow engine.

##### 3.1.1 User Authentication

To be able to use an ORKA enabled application, a business user needs an ORKA user account in the Policy Storage Service (PSS). The user has to log on at the Policy Decision Point (PDP) before he can use the application. The PEP takes the user id and the user-supplied password and forwards it to the PDP.

The PDP will check if the user and password are valid. If so, the PDP will read the user’s roles and permissions and create a session certificate. The identifier of the certificate will be returned to the PEP who stores it for later access requests being sent to the PDP.

The session certificate may have a certain validity (some minutes) and can be cached by the PDP to avoid database operations on each access request. If the session request expires, the PDP will transparently issue a new session certificate.

### **3.1.2 Application Policy Enforcement**

To make use of ORKA security functions in applications, ORKA provides a function library interface and a web service interface. These interfaces act as a PEP and forward access requests to the PDP.

An organization using ORKA can use these interfaces if it can modify the application itself, e.g. inserting access decision request code into the source code.

### **3.1.3 Workflow Policy Enforcement**

Workflow applications can call ORKA functions either directly by calling ORKA interface functions, or by using the ORKA callback PEP. This PEP is integrated into the workflow engine and is automatically being called if certain workflow events occur. Using this callback, the workflow security policy can be enforced without any changes to the workflow application itself.

The ORKA callback PEP for use in workflow engines will be developed in work package 1.7.

### **3.1.4 Legacy System Policy Enforcement**

Business applications that neither have any source code that can be modified nor expose any (callback) interfaces to integrate access decision functions demand a different strategy to enforce ORKA security policies.

For these legacy applications, the ORKA PDP transforms the security policy specified in the ORKA Policy Language (OPL) into a policy that can be understood by the system. This transformation will be done on publishing the OPL policy (see the Policy Administration Point section for further details).

As a pitfall, it is sure that some features of the ORKA policy language cannot be translated into a legacy security policy.

Another problem is that ORKA does not have any control about policy enforcement as this is done autonomously by the external system. Therefore, ORKA cannot easily get access to audit trails and other security-relevant information.

As a result, there exists an impedance mismatch between the policy models and less secure operation of the legacy system compared to the ORKA-controlled application.

## **3.2 Policy Decision Point (PDP)**

This section describes the business functions of the ORKA policy decision point (PDP). The PDP evaluates access decision requests received from the PEP and returns the decision result.

### **3.2.1 Policy Object Selection**

Upon receiving an access decision request from the PEP, the PDP looks up the composite policy object that is responsible to handle requests for the target and action specified in the request.

If the policy object has not been loaded, it will be read from the Policy Storage Service (PSS). After loading the object successfully, the policy object will be cached for further requests to avoid heavy database load. After some minutes, the policy object expires and will be retrieved again if another request comes in.

### **3.2.2 Policy Translation**

After the policy object has been loaded from the PSS, it has to be translated from the ORKA Policy Language (OPL) into the language of the enforcement system. As ORKA uses XACML for enforcement, the policy object OPL information will be converted to XACML and passed to the enforcement kernel which is a wrapper around the SUN XACML reference implementation.

This type of policy translation occurs only for policies that are actively enforced by the ORKA PDP. Policies meant to be used in legacy systems will be translated to a legacy policy upon policy deployment. See the Policy Administration Point (PAP) section in work package 5.3 for further details.

### **3.2.3 Policy Validation**

The Policy Decision Point (PDP) does not perform policy object validation. It assumes that the policy object is validated and correct.

The ORKA Policy Administration Point (PAP) ensures that a policy has to be validated before it can be deployed. As the PDP only uses deployed policies, an invalid policy cannot be used for access decision operations.

The policy validation operations are described in work package 5.3.

## **3.3 Policy Storage Service (PSS)**

The Policy Storage Service (PSS) contains all policy and master data objects that are used by the PDP.

The PSS structure and interfaces are described in work package 5.3.

## **3.4 Policy Context Service (PCS)**

The policy objects used for access decision requests may contain references to context information or other information.

If a policy object is translated from OPL to the enforcement language, these context references will remain untranslated. Context references are resolved dynamically as access decision requests are processed.

Therefore, the implementation of the enforcement language must support callbacks for resolving context attributes. In the ORKA reference implementation, we use XACML-Finders that are deployed in addition to the SUN reference implementation, connecting the XACML enforcer with external ORKA-based context information.

## 4 Programming Guidelines

The ORKA project has defined some programming guidelines to be able to have a high standard in software development. Special attention has been paid to readability and maintainability of the code base.

ORKA uses the Java programming language for the implementation of the prototype. Using Java, there are multiple components available that make the implementation of the prototype easier. In general, it is possible to port the system architecture to a different programming language as no language-specific features will be used.

### 4.1 Repository Directory Structure

The ORKA project has defined a common directory structure in the central source code repository (see 7).

The folder *documents* contains articles and information concerning software development.

The folder *examples* contains ready-to-run examples of technologies used in the project, e.g. JBoss Seam and Drools.

The folder *implementation* contains the Eclipse workspaces of different project releases. The first project release is *orka-release-0*, other releases may be defined later.

### 4.2 Software Development Environment

The ORKA software will be implemented in the Java programming language. For development and testing, the development system Eclipse will be used in version 3.3 (Europa). The plugin Subclipse is used to connect the distributed developers in the project with the central ORKA software repository.

As multiple developers are working in the project, a common set of code conventions has to be defined. ORKA used pre-configured Eclipse formatting settings and the free plugin Jalopy to enable simple, automated source code formatting.

The configuration of the Eclipse workspace for use in ORKA projects has been described in a Developers Handbook (available in German on the BSCW server).

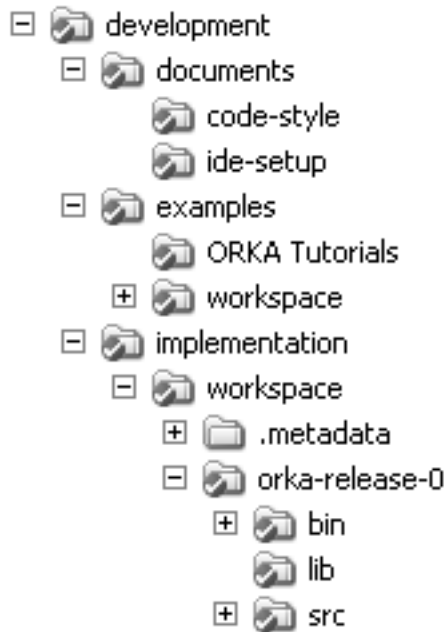


Figure 7: Development Directory Structure

### 4.3 Java Package Structure

This section describes the basic Java package structure. Further packages may be defined during implementation of the ORKA prototype.

- **org.orka.admin**  
Base package of ADMIN work packages.
- **org.orka.admin.db**  
Administrative database classes.
- **org.orka.admin.interfaces**  
Administrative interface classes.
- **org.orka.enforce**  
Base package of ENFORCE work packages.
- **org.orka.enforce.context**  
Classes related to context information processing.
- **org.orka.enforce.db**  
Enforcement database classes.
- **org.orka.enforce.interfaces**  
Enforcement interfaces.
- **org.orka.enforce.lang.drools**  
Base package for DROOLS policy enforcement.
- **org.orka.enforce.lang.xacml**  
Base package for XACML policy enforcement.

- **org.orka.opl**  
Base package for all OPL-related classes.
- **org.orka.enforce.db**  
Enforce-related database classes.
- **org.orka.test**  
Base package for test applications.
- **org.orka.test.opl**  
Test applications for OPL.
- **org.orka.valid**  
Base package of VALID work packages.
- **org.orka.valid.interfaces**  
Validation-related interface classes.

## 5 Policy Enforcement Prototype

This chapter describes the implementation of the policy enforcement prototype. The prototype uses the XACML language for policy decision. The ORKA security policies defined in OPL (ORKA Policy Language) will be translated to XACML. The exact translation will be part of wworkpackage 2.6. In this workpackage we give an overview about XACML and show the parts relevant for the enforcement component as well as which parts of the XACML policies are relevant or context information resolution.

### 5.1 XACML

To understand the example it is crucial to know how XACML works. We will give a brief overview about the main elements in XACML in the following paragraphs. A complete introduction into XACML and its specification, however, is out of scope for this deliverable. The reader may be referred to the specification [Mos05] and other XACML literature ([LPL<sup>+</sup>03], [MBCS06]).

Access control by using XACML relies on a typical request - response architecture. Whenever a subject requests access for a certain resource with a given action an access request is generated. This request is sent to an access request evaluation component which evaluates it based on pre-defined policies. When evaluated the component returns the result as a response to the sender. The requester then, acting as enforcement point, must reliably enforce the returned decision which means the application either denies or permits the requested action on the requested resource. In the following paragraph we will describe the architecture, a typical XACML request, XACML policy and XACML response in more detail.

### 5.1.1 Request - Response Architecture

A key element in every access control architecture is the interaction between enforcement component and decision component. The enforcement component is usually placed within the application and must reliably enforce permit or deny decisions, made by the decision component. The basic idea behind an enforcement point is a simple if-then construct. Whenever a certain action should be performed on a specific resource a request is generated, it is sent to an access evaluation component and if this component returns permit, the requested action is executed, otherwise the action is not executed and possibly some exception handling is performed.

Clearly, it is important that the requester and the actual decision component understand each other. For this reason XACML specifies XML-based request and response messages. The enforcement component (called Policy Enforcement Point, PEP) is responsible for generating the request message, sending it to the evaluation component, and translating the response to a format the application internal enforcement mechanisms (i.e., the above mentioned if-then construct) understands.

### 5.1.2 XACML Request

A request consists of the following information:

- **Subject ID:** identifies the subject (e.g., the user) which requests access, contains possibly additional attribute information (attribute values) about the subject (e.g., role memberships)
- **Resource ID:** identifies the resource a subject wants to access, contains possibly additional attribute information (attribute values) about the resource (e.g., process instance ID, task instance ID, ...)
- **Action:** specifies the action the subject wants to perform on the given resource contains possibly additional attribute information (attribute values) about the action (e.g., transfer parameters for a method call)

An exemplary request for the above mentioned banking process might look like the XML document given next.

Listing 1: XACML: Example Authorization Request

```
1 <Request>
2   <Subject>
3     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
4       DataType="http://www.w3.org/2001/XMLSchema#string">
5       <AttributeValue>Max Mayer</AttributeValue>
6     </Attribute>
7     <Attribute AttributeId="role"
8       DataType="http://www.w3.org/2001/XMLSchema#string"
9       Issuer="ContextHandler">
10    <AttributeValue>Employee</AttributeValue>
11    <AttributeValue>PreProcessingClerk</AttributeValue>
12  </Attribute>
13 </Subject>
```

```

14 <Resource>
15   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
16     DataType="http://www.w3.org/2001/XMLSchema#string">
17     <AttributeValue>LoanOriginationProcess.InputCustomerData</AttributeValue>
18   </Attribute>
19   <Attribute AttributeId="ProcessInstanceId"
20     DataType="http://www.w3.org/2001/XMLSchema#integer">
21     <AttributeValue>12345</AttributeValue>
22   </Attribute>
23   <Attribute AttributeId="TaskInstanceId"
24     DataType="http://www.w3.org/2001/XMLSchema#integer">
25     <AttributeValue>67890</AttributeValue>
26   </Attribute>
27 </Resource>
28 <Action>
29   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
30     DataType="http://www.w3.org/2001/XMLSchema#string">
31     <AttributeValue>Assign</AttributeValue>
32   </Attribute>
33 </Action>
34 </Request>

```

In this example, the section for the subject contains the name of the user requesting access "Max Mayer" as well as the roles he is member of "Employee" and "PreProcessingClerk".

Further, the Resource section contains the resource the user wants to access which is the first task in the banking process with its full qualified name "LoanOriginationProcess.InputCustomerData". Additionally the requests contains the identifiers for this specific process and task instance.

The Action section contains the method the user wants to perform. In this case it is an "Assign"-action which is used by a user to claim a task for his or her worklist as we will explain later.

### 5.1.3 XACML Response

The response consists of the following information:

- **Decision:** Specifies the evaluated result for a request which can be any of the following options: Permit, Deny, Indeterminate, NotApplicable.
- **Status Code:** This field is optional. It indicates errors during the evaluation and gives optionally information about them.
- **Obligation:** This element might contain a list of obligations which have to be performed by the PEP when receiving the response. If one of the obligations cannot be fulfilled, the PEP must act as if the request would have been denied. (For further information, please be referred to the specification [Mos05].)

A typical response message might look like the XML document given next.

Listing 2: XACML: Example Authorization Response

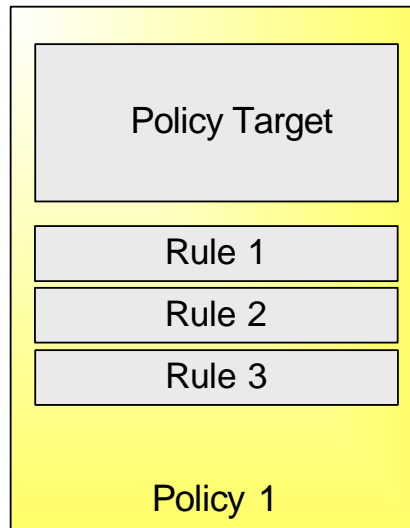


Figure 8: XACML Policy

```

1 <Response>
2   <Result>
3     <Decision>\textbf{Permit}</Decision>
4     <Status>
5       <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok" />
6     </Status>
7   </Result>
8 </Response>

```

#### 5.1.4 XACML Policy Structure

An single XACML Policy is generally structured by a Policy Target-element and at least one but possibly several Policy Rule-elements. An abstract illustration is given with Figure 8.

The target specifies for which type of access control requests this particular policy is applicable. This means for a particular type of request this policy provides an answer. If the target matches the request, one of the rules within the policy should be able to return whether access should be granted or denied.

Assume, for instance, the Policy target specifies "Subject == Max Mayer" and "Resource == LoanOriginationProcess.InputCustomerData". In this case for all requests having Max Mayer as subject and the first task of our Banking Process as resource, this policy would be processed by the PDC to evaluate whether Max is permitted to access the task. In other words, this policy provides answers for requests concerning Max Mayer and the InputCustomerData-task of the banking process. For any other request, this policy is ignored.

The rule of a policy specifies the core of every Policy. A rule defines the requirements for a permission and its effect. A requirement might be that the request must be for the "Action == Assign" in which case a certain effect applies (e.g., PERMIT). This means, whenever this rule is analyzed (and this is the case whenever the complete policy is a target for the request) and the

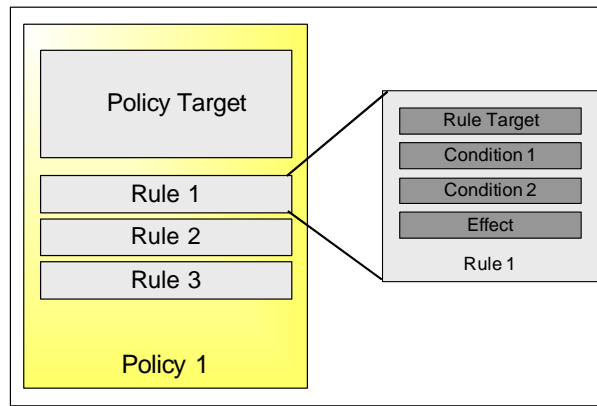


Figure 9: XACML Policy with detailed Rules

analysis results that the requirements are fulfilled, the effect is returned. An effect may either be PERMIT or DENY.

A more complex requirement may include one or more conditions such as "Time() >= 6 a.m. && Time() <= 5 p.m." which would mean that additionally to the requirement "Action == Assign" the system time (for illustration represented by the function Time()) must be between 6 a.m. and 5 p.m.

More concrete, as illustrated in Figure 9 every rule has a target for itself (which defines the above mentioned requirement for the permission and its effect). For those requests that match with the rule target (similar to the policy target) the rule's conditions are evaluated. If the conditions evaluate to true, the value of the Effect-element is returned. If one of the conditions does not evaluate to true (i.e., remains not satisfied) nothing is returned and the evaluation process continues with the next rule.

Summarizing, the mechanism of targets in the policy define for which types of request a certain policy contains rules that specify whether the response for the request should be PERMIT or DENY; the targets and conditions of a rule specify for which types of requests and other conditional facts (e.g., the system time) the rule returns its defined effect (i.e., either PERMIT or DENY).

For example, the whole policy can be the target for a whole group of subjects (e.g., all members of the role PreProcessingClerk) and resource (e.g., LoanOriginationProcess.InputCustomerData). And every rule in the policy specifies the conditions and effects for a certain action (e.g., Action == Assign: Effect == PERMIT, Action == Cancel: Effect == DENY).

Policies can be grouped into "Policy Sets". Each Policy Set contains at least one but possibly an unbounded number of different policies. Further, it specifies a "Policy Set Target" element which defines analogous to a policy or a rule for which requests the set applies. See Figure 10 for illustration.

A sample policy in XACML is given with Listing 3. It defines a user must be member of the role "PreProcessingClerk" to be assigned to the first task "Input Customer Data" of the the Banking Process described in the work package 1.1. The listing is meant as example to illustrate the explained structure of XACML policies. It is not complete, yet. Further aspects will be added in the following sections.

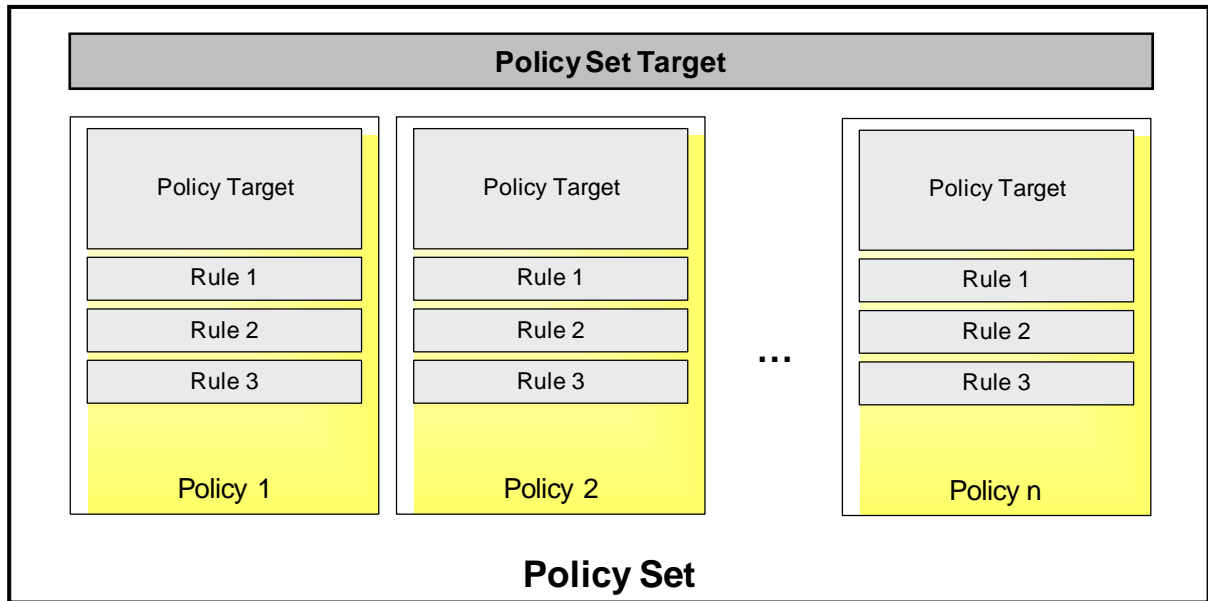


Figure 10: XACML Policy Set

Listing 3: XACML: Example Authorization Policy

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy
3   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy-cs-xacml-schema-policy-01.xsd"
6   PolicyId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:task01"
7   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
8   >
9   <Description>
10    Example Policy for the Loan Origination Process, Task Input Customer Data.
11    The policy is for illustration and does not cover all aspects.
12    An extended version of this policy for the banking process follows in a later section.
13  </Description>
14  <Target>
15    <Subjects>
16      <AnySubject/>
17    </Subjects>
18    <Resources>
19      <Resource>
20        <ResourceMatch
21          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
22          <AttributeValue
23            DataType="http://www.w3.org/2001/XMLSchema#string">
24            LoanOriginationProcess.InputCustomerData</AttributeValue>
25          <ResourceAttributeDesignator
26            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
27            DataType="http://www.w3.org/2001/XMLSchema#string"/>
28          </ResourceMatch>
29        </Resource>
30      </Resources>
31    <Actions>
32      <AnyAction/>
33    </Actions>
34  </Target>

```

```

33 <Rule
34     RuleId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:task01:rule01"
35     Effect="Permit">
36     <Description>
37         A member of "PreProcessingClerk" can be assigned to the task (Action == "Assign").
38     </Description>
39     <Target>
40         <Subjects>
41             <Subject>
42                 <SubjectMatch
43                     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
44                     <AttributeValue
45                         DataType="http://www.w3.org/2001/XMLSchema#string">
46                         PreProcessingClerk</AttributeValue>
47                     <SubjectAttributeDesignator
48                         AttributeId="role"
49                         DataType="http://www.w3.org/2001/XMLSchema#string" />
50                     </SubjectMatch>
51                 </Subject>
52             </Subjects>
53             <Resources>
54                 <AnyResource/>
55             </Resources>
56             <Actions>
57                 <Action>
58                     <ActionMatch
59                         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
60                         <AttributeValue
61                             DataType="http://www.w3.org/2001/XMLSchema#string">Assign</
62                             AttributeValue>
63                         <ActionAttributeDesignator
64                             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
65                             DataType="http://www.w3.org/2001/XMLSchema#string" />
66                         </ActionMatch>
67                     </Action>
68                 </Actions>
69             </Target>
70     </Rule>
71 </Policy>

```

## 5.2 XACML Use Case Policy Example

This section completes the above example for of the policy for task one "Input Customer Data" and gives another policy example for a second tasks which relies on additional history data to be provided by the Context Information Service. The examples will continue to be in the policy language XACML as for the demonstrator developed within this project one of the enforcement components and the respective decision component will be based on XACML. (For the second decision component relying on DROOLS, a separate workpackage 3.5 describes the accompanying policy structure and decision evaluation.)

As described in workpackage 1.6, the task life cycle within JBoss is given by Figure 11. The actions a user is able to perform to access a task are also given in the figure and summarized as

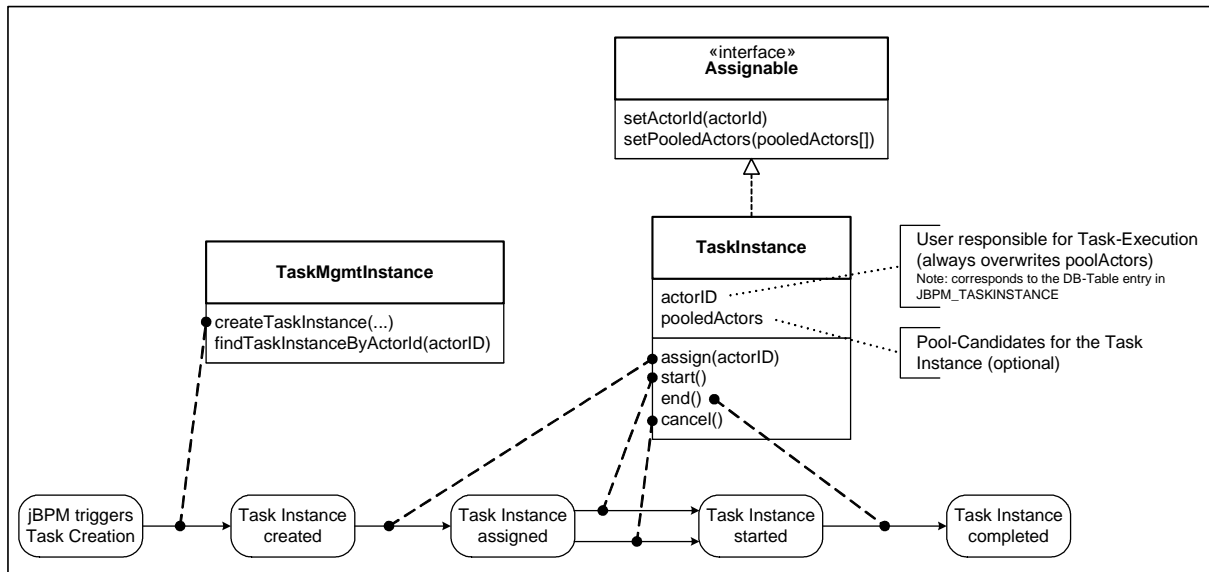


Figure 11: Task Life Cycle within JBoss

follows.

- **Claim a task:** assign() or setActorId()
- **Start a task:** start()
- **Cancel a task:** cancel()
- **Finish a task:** end()

These method calls are secured within the workflow engine which means whenever one of these methods are called by the engine, a security request is generated and sent to the PDP for evaluation. The request contains one of the above methods as 'action', the task for which it is called as 'resource', and the user who wants to call the method as 'subject'.

The PDP (respectively the PDC) has to be able to evaluate the a request using the stored policies. Hence, the above methods have to be covered by security policies for every workflow definition which might run on the engine. So the first step to define the policies for the banking process in XACML is to define XACML rules for the above methods for each task of the process.

We complete the respective part of the policy for the first task in the banking process. We add the additional actions start, end, and cancel to the rule's target to show specify that a user can do each of these actions on the task. Listing 4 shows the changed part.

A second change the listing shows affects the role resolution. We use the so called SubjectAttributeDesignator to resolve the roles for which a subject is member. As the request-context has to provide this information, the context manager requires the location where the information is stored. Responsible for context information resolution is the context information service (CIS). This service receives a context request which consists of a string which defines what information is needed. In our case the string is "Logon.Subject.Roles" as we need the roles of the current Subject. The string resolution is done by the CIS. Therefore, the SubjectAttributeDesignator-element within an XACML policy provides this string which is passed to the CIS.

Listing 4: Authorization Policy for Task 1: Input Customer Data of the Banking Process Case Study given in Workpackage 1.1

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy
3     xmlns="urn:oasis:names:tc:xacml:1.0:policy"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy-cs-xacml-schema-policy-01.xsd"
6     PolicyId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:task01"
7     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
8     >
9     <Description>
10        Example Policy for the Loan Origination Process, Task Input Customer Data.
11        The policy is for illustration and does not cover all aspects.
12        An extended version of this policy for the banking process follows in a later section.
13    </Description>
14    <Target>
15        <Subjects>
16            <AnySubject/>
17        </Subjects>
18        <Resources>
19            <Resource>
20                <ResourceMatch
21                    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
22                    <AttributeValue
23                        DataType="http://www.w3.org/2001/XMLSchema#string">
24                        LoanOriginationProcess.InputCustomerData</AttributeValue>
25                    </ResourceAttributeDesignator
26                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
27                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
28                </ResourceMatch>
29            </Resource>
30        </Resources>
31        <Actions>
32            <AnyAction/>
33        </Actions>
34    </Target>
35    <Rule
36        RuleId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:task01:rule01"
37        Effect="Permit">
38        <Description>
39            A member of PreProcessingClerk can be assigned to the task.
40            Additionally, a member can also start, end, and cancel the task.
41            The SubjectAttributeDesignator contains a string, where to find the role
42            information.
43        </Description>
44        <Target>
45            <Subjects>
46                <Subject>
47                    <SubjectMatch
48                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
49                            <AttributeValue
50                                DataType="http://www.w3.org/2001/XMLSchema#string">
51                                    PreProcessingClerk</AttributeValue>
52                            <SubjectAttributeDesignator
53                                AttributeId="Logon.Subject.Roles"
54                                DataType="http://www.w3.org/2001/XMLSchema#string"/>
55                        </SubjectMatch>
56                    </Subject>
57                </Subjects>

```

```

54     <Resources>
55         <AnyResource/>
56     </Resources>
57     <Actions>
58         <Action>
59             <ActionMatch
60                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
61                 <AttributeValue
62                     DataType="http://www.w3.org/2001/XMLSchema#string">Assign</
63                     AttributeValue>
64                 <ActionAttributeDesignator
65                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
66                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
67             </ActionMatch>
68         </Action>
69         <Action>
70             <ActionMatch
71                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
72                 <AttributeValue
73                     DataType="http://www.w3.org/2001/XMLSchema#string">Start</
74                     AttributeValue>
75                 <ActionAttributeDesignator
76                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
77                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
78             </ActionMatch>
79         </Action>
80         <Action>
81             <ActionMatch
82                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
83                 <AttributeValue
84                     DataType="http://www.w3.org/2001/XMLSchema#string">End</
85                     AttributeValue>
86                 <ActionAttributeDesignator
87                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
88                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
89             </ActionMatch>
90         </Action>
91         <Action>
92             <ActionMatch
93                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
94                 <AttributeValue
95                     DataType="http://www.w3.org/2001/XMLSchema#string">Cancel</
96                     AttributeValue>
97                 <ActionAttributeDesignator
98                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
99                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
100             </ActionMatch>
101         </Action>
102     </Actions>
103 </Target>
104 </Rule>
105 </Policy>

```

To give a further example with respect to separation of duty. A policy's target can be defined to be relevant only for a certain task. Further it can be defined within this policy that if this task is requested for execution in a users task history (the tasks a user completed) cer-

tain other tasks may not appear; otherwise access is denied. This is done using conditions. The following listing 5 shows such how such a condition may look like. It conditions that a certain task "ExclusiveTask" may not be in the set of history tasks of a user, given that the access request is made for the target "Process.CurrentRequestedTask". To retrieve the users completed tasks such that they are available for comparison within the XACML request context, the following string is used for the AttributeDesignator to be passed on to the CIS: "Logon.Subject.Process[getId].getListOfExecutedTasks"

Listing 5: XACML-based SoD condition (policy excerpt)

```

1 [...]
2 <Rule>
3   <Target>
4     [...]
5     <Resources>
6       <Resource>
7         <ResourceMatch
8           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
9           <AttributeValue
10            DataType="http://www.w3.org/2001/XMLSchema#string">Process.
11              CurrentRequestedTask</AttributeValue>
12           <ResourceAttributeDesignator
13             AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
14             DataType="http://www.w3.org/2001/XMLSchema#string"/>
15           </ResourceMatch>
16         </Resource>
17       </Resources>
18     </Target>
19     <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
20       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
21         <AttributeValue
22           DataType="http://www.w3.org/2001/XMLSchema#string">ExclusiveTaskName</
23           AttributeValue>
24         <SubjectAttributeDesignator
25           DataType="http://www.w3.org/2001/XMLSchema#string"
26           AttributeId="Logon.Subject.Process[getId].getListOfExecutedTasks"/>
27       </Apply>
28     </Condition>
29 </Rule>
30 [...]
```

Further, every task in a process might call back-end methods provided by the business objects or web services the task accesses. The Case Study describing the Banking Process in work package 1.1 especially shows that there are explicit method calls on business objects behind every task. In other words, whenever a user executes task 1 ("Input of Customer Data") she must eventually be able to store the entered data by calling an method of a back-end business object to which the data is transferred and processed. The business object responsible for storing the customer data for the given example is called "Customer Data" as Table 1 in work package 1.1 shows.

For this we add a new policy which solely specifies the access information for the business object "Customer Data". The policy is given with Listing 6.

Listing 6: XACML: Authorization Policy specifying access for the Business Object Customer Data

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <Policy
3     xmlns="urn:oasis:names:tc:xacml:1.0:policy"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy-cs-xacml-schema-policy-01.xsd"
6     PolicyId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:customer-data"
7     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
8     >
9     <Description>
10        Example Policy for the Loan Origination Process, Business Object Customer Data.
11    </Description>
12    <Target>
13        <Subjects>
14            <AnySubject/>
15        </Subjects>
16        <Resources>
17            <Resource>
18                <ResourceMatch
19                    MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
20                    <AttributeValue
21                        DataType="http://www.w3.org/2001/XMLSchema#string">CustomerData</
22                        AttributeValue>
23                    <ResourceAttributeDesignator
24                        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
25                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
26                </ResourceMatch>
27            </Resource>
28        </Resources>
29        <Actions>
30            <AnyAction/>
31        </Actions>
32    </Target>
33    <Rule
34        RuleId="urn:oasis:names:tc:xacml:1.0:banking-process:policy:customer-data:rule01"
35        Effect="Permit">
36        <Description>
37            A User of role PreProcessingClerk may perform the actions query and update
38            on the Business Object Customer Data.
39        </Description>
40        <Target>
41            <Subjects>
42                <Subject>
43                    <SubjectMatch
44                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
45                            <AttributeValue
46                                DataType="http://www.w3.org/2001/XMLSchema#string">
47                                PreProcessingClerk</AttributeValue>
48                            <SubjectAttributeDesignator
49                                AttributeId="Logon.Subject.Roles"
50                                DataType="http://www.w3.org/2001/XMLSchema#string"/>
51                        </SubjectMatch>
52                    </Subject>
53                </Subjects>
54            <Resources>
55                <AnyResource/>
56            </Resources>
57            <Actions>

```

```

55         <Action>
56             <ActionMatch
57                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
58                 <AttributeValue
59                     DataType="http://www.w3.org/2001/XMLSchema#string">Query</
60                     AttributeValue>
61                 <ActionAttributeDesignator
62                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
63                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
64             </ActionMatch>
65             <ActionMatch
66                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
67                 <AttributeValue
68                     DataType="http://www.w3.org/2001/XMLSchema#string">Update</
69                     AttributeValue>
70                 <ActionAttributeDesignator
71                     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
72                     DataType="http://www.w3.org/2001/XMLSchema#string"/>
73             </ActionMatch>
74         </Action>
75     </Actions>
76 </Target>
77 </Rule>
78 </Policy>

```

### 5.3 Runtime Environment

ORKA uses the programming language Java and different related software components. The main software components required to run the ORKA prototype are:

- **Java Runtime Environment (JRE), Release 5.0**  
The runtime environment is needed to execute Java programs.
- **JBoss Application Server**  
The JBoss application server is the central component that contains the ORKA services.
- **JBPM Workflow Engine**  
The JBPM workflow engine contains the process definitions used in the ORKA case studies.
- **JBoss Rules Engine (DROOLS)**  
The rule engine is used to implement separation of duty constraints.
- **Hypersonic database**  
Hypersonic is the database system used to store policy and user information.

The software versions of the the components can be found in the ORKA installation guide.

## 5.4 Software Testing

ORKA uses unit and system testing for software quality assurance. Furthermore, test applications will be defined to test and illustrate specific aspects of the ORKA architecture.

The work packages can create additional packages under the base Java package *org.orka.test*. The package *org.orka.test.simple*, as an example, contains a simple PEP/PDP implementation of a policy enforcement engine.

Procedures for user interface quality assurance will be defined in the ADMIN work packages.

## 6 Conclusion

In this work package we have described the first prototype of the ORKA policy enforcement components. The implementation of the basic prototype has started.

In chapter 2 we have given an overview about the platform architecture of the policy enforcement components. Certain attention has been put on the design of the security architecture and platform environment. To examine the security architecture we have had a close look at authentication and access control technologies, at role based access control and at business application integration. In the platform environment, we have designed network communication protocols, cryptographic functions and important log and audit functions.

In chapter 3 the components of the platform services have been described. The Policy Enforcement Point (PEP) with user authentication, application policy enforcement, workflow policy enforcement and legacy system policy enforcement has been designed. We defined the Policy Decision Point (PDP) with policy object selection, policy translation and policy validation. Furthermore, the Policy Storage Service (PSS) and the Policy Context Service (PCS) have been mentioned. They will be defined in detail in work package 5.3.

Before the implementation starts it is very important to define the programming guidelines to be used in the ORKA project. The repository directory structure, the development environment and the Java package structure have been explained in chapter 4.

The ORKA project partners have decided to use XACML (eXtensible Access Control Markup Language) as the policy enforcement language. In chapter 5 we have given a first introduction towards XACML with some examples related to the ORKA banking process case study from work package 1.1.

The initial prototype implementation is stored in the ORKA subversion directories.

In work package 3.4 we will extend the prototype implementation and continue our work in the enforcement area.

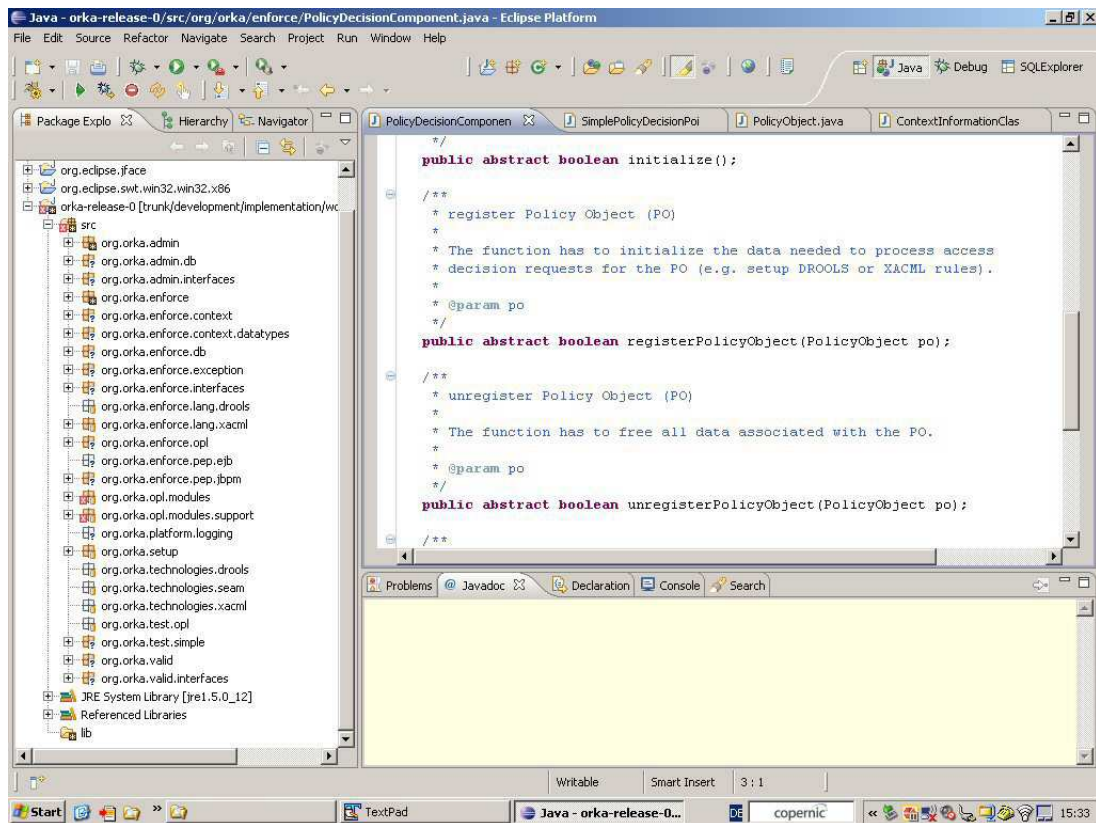


Figure 12: First ORKA prototype implementation

## References

- [LPL<sup>+</sup>03] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using xacml for access control in distributed systems. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 25–37, New York, NY, USA, 2003. ACM.
- [MBCS06] P. Mazzoleni, E. Bertino, B. Crispo, and S. Sivasubramanian. Xacml policy integration algorithms: not to be confused with xacml policy combination algorithms! In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 219–227, New York, NY, USA, 2006. ACM.
- [Mos05] Tim Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Standard.