SEVENTH FRAMEWORK PROGRAMME
Challenge 1
Information and Communication Technologies

Trusted Architecture for Securely Shared Services

Document Type: Software Deliverable

Title: # Trust Tool Set

Work Package: WP5

Deliverable Nr: D5.4

Dissemination: Public

Preparation Date: June 27, 2011

Version: v3.0

The TAS³ Consortium

	Beneficiary Name	Country	Short	Role
1	KU Leuven	BE	KUL	Coordinator
2	Synergetics NV/SA	BE	SYN	Partner
3	University of Kent	UK	KENT	Partner
4	University of Karlsruhe	DE	KARL	Partner
5	Technische Universiteit Eindhoven	NL	TUE	Partner
6	CNR/ISTI	IT	CNR	Partner
7	University of Koblenz-Landau	DE	UNIKOL	Partner
8	Vrije Universiteit Brussel	BE	VUB	Partner
9	University of Zaragoza	ES	UNIZAR	Partner
10	University of Nottingham	UK	NOT	Partner
11	SAP Research	DE	SAP	Project Mgr
12	EIfEL	FR	EIF	Partner
13	Intalio	UK	INT	Partner
14	Risaris	IR	RIS	Partner
15	Kenteq	NL	KETQ	Partner
16	Oracle	UK	ORACLE	Partner
17	Custodix	BE	CUS	Partner
18	Medisoft	NL	MEDI	Partner
19	Symlabs	PT	SYM	Partner

Contributors

	Name	Organisation
1	Jerry den Hartog	TUE
2	Christian Hütter	KARL
3	Slim Trabelsi	SAP
4	Stijn Lievens	KENT
5	Elisa Costante	TUE

Contents

1 Executive Summary

The TAS³ Trust Policy Management architecture [D5.1] consists of a collection trust services with the Trust PDP providing the integration of and interface to these services. This third iteration of the implementation of this architecture consists of the trust PDP, three distinct trust services, a reputation based trust (RTM) service a credential based trust (CTM) service, and an online compliance testing trust (OCTTM) service and supporting tools. This document focuses on the Trust PDP, CTM service, OCTTM service and tools with [D5.2] describing the RTM service. The updates in this third iteration focus on improving ease of deployment of Trust PDP, integration of updated trust services, interface updates and support for controlling the order of evaluation for nested trust metrics through the call back mechanism. A developer tutorial describing the updated template for trust services and providing technical details of the implementation for the benefit of developers of additional trust services has been added.

The Trust PDP is Java based, accepts XACML [XACML] request context objects, evaluates trust policies embedded in XACML-style XML wrappers and returns standard XACML permit/deny responses. The Trust PDP enables the authentication/authorization framework to incorporate trustworthiness of requesters in their access decisions. The Trust PDP offers support for obligations and break the glass mechanism [D7.1] allowing the use of these features within trust policies as well.

A web-service interface to the (Java based) Trust PDP is provided by integrating it in the Standalone Authorization Server software package provided by the University of Kent, guaranteeing an equal WSDL support/interpretation of the SOAP messages.

The Trust Information Access Service Provider is a helper component which offers e.g. the Service Discovery access to raw trust score information. It accepts the same requests as the Trust PDP but, in addition to a permit/deny response it also provides a trust ranking according to a specific trust metric. This allows service discovery to offer trusted services to the user sorted by their score on the trust metric.

The CTM service provides trust metric based on credential (chains). The POLIPO trust management system [TSZE09a, TSZE09b] forms the basis of this service. The service offers a SAML [SAML] compatible interface and uses SAML assertions to encode trust credentials. The updated CTM service performs caching of non-local credentials itself, removing the need for the credential cache implemented in the previous iteration of the Trust PDP.

The online compliance testing based trust service (OCTTM) provides trust metrics based on the quality (policy compliance) of services. This service monitors the audit bus (component T3-BUS-AUD [D2.1]) for posted OCT [D10.3] results.

1.1 Reading Guide

The next chapter describes the functionality of the software and its place in the trust architecture and the role in the overall TAS³ architecture and the limitations of this third iteration of the software in general terms. Chapter 3 provides installation and

configuration information needed to get the software up and running. Chapter 4 describes how to use the software.

Chapters 5-8 provide a detailed description of the inner working of the software for developers looking to code against and interface with these components. Chapter 5 provides the software architecture, Chapter 6 its interfaces (APIs), Chapter 7 the BSD style license under which the code is released and Chapter 8 conclusions and further work.

2 Introduction to the Software

2.1 Purpose

The main purpose of the trust tool set is to enable trust evaluation based on different sources of trust information within the TAS³ architecture. To this end a trust tool set is created implementing a standardized XACML style interface to trust evaluation for resource access requests and a number of different trust services specialized to gathering and processing trust information of a specific type as well as supporting components.

2.2 Scope

This document covers the trust tool set; the Trust PDP (T3-PDP-T) with trust service client software, the Trust Information access service provider (T3-TRU-SP) and the quality based (OCTTM) trust service. The RTM service is described in a dedicated deliverable [D5.2] and the credential based trust (CTM) service (T3-TRU-CTM) in the deliverable on novel trust metrics [D5.3].

2.3 Functionality

This is the third iteration of the trust tool set.

Several possible levels of policy languages were introduced at a high level of abstraction in [D5.1]. The implementation of the trust policy language supported by the current Trust PDP (see Section 5.3) uses trust functions to incorporate trust metrics in a XML document adhering to the XACML policy standard. This approach enables expressing level 1, 2 and level 3 policies [D5.1]. Evaluation of the trust policies is supported for 1, 2 and limited level 3 policies. In the current implementation specific support is still needed from the trust function to enable embedding implying some limitations to the generality of the support for level 3 policies also, the trust PDP callback option, needed for more intricate interaction in the evaluation of level 3 policies is not yet supported in this iteration.

The current Trust PDP implementation has support for three services built in though it can support any number of trust services. Adding an additional trust service in the trust PDP requires implementing a web service call wrapper in the form of a function (See Section 5.3.1.2) and adding an entry to the trust service configuration file. The three services are a reputation based trust (RTM) service, a credential based trust (CTM) service, and a quality (according to online compliance tests) based trust (OCTTM) service. Additionally several convenience and testing oriented services such as a service offering generic trust functions, including a `trust everybody' trust function, and a trust function simulation service are included.

The OCTTM service enables trust decisions on the quality of a website. The OCT [D10.3] tests determine whether a service adheres to its stated policies. Thus the pass/fail rate on these tests provides an estimate of the quality of a service which is an important factor in a user's trust in a service. The OCTTM service gathers test results by listening to the audit bus [D2.1] for posted test results. Several trust metrics extract trust levels from the overall testing evidence collected in this way. A basic

metric checks for presence of tests. Though basic, a tested service may be more trustworthy that an untested one even if not all tests are passed. A second metric provides a more detailed view of quality by considering the ratio of pass/fail for the available tests. More intricate metrics which not only consider the ratio of passed test but also the importance of each passed or failed test are possible but not included in the current implementation; a classification of different tests and their importance to trust is needed for this.

The CTM service described in [D5.3] implements the POLIPO framework [POLIPO, TSZE09a, TSZE09b]. The RTM service is described in detail in [D5.2]. In this document we focus on the connectors used to embed these services in the trust framework rather than this service itself.

2.4 Available Releases and Components

Component T3-PDP-T: Trust PDP

Version	Description	Notes
0.1	First release to the project of the Trust PDP.	Defines interfaces but not stable.
0.2	Updated release with Trust PDP supporting both CTM and RTM services.	Stable but only provides a Java interface; not yet web service. Starting point integration with Kent's standalone authorization server.
0.3	Dependency cleanup of Trust PDP. Supports web based Trust PDP.	Starting point integration in demonstrators.
0.4	Updated release based on feedback demonstrators.	Version integrated into year 2 demonstrators.
0.5	Update with new services, changed basic trust service structure, sticky policy support.	Clean up of code, easier to use trust service template and added functionality. Starting point integration year 3 demonstrators.
1.0	Updated with new services, based on updated template.	Improved service template, reducing effort needed to support new metrics/services. Support call-back mechanism. Starting point for further exploitation.

Component T3-TRU-SP

Version	Description	Notes
0.1	First release to the project of the Trust service discovery helper.	Starting point integration in year 2 demonstrators.
0.5	Update based on T3-PDP-T v0.5 code.	Starting point integration in year 3 demonstrators
1.0	Update based on T3-PDP-T v1.0 code.	Support for dynamic querying of trust metrics. Starting point for further exploitation.

3 Installation Guidelines

3.1 Hardware and Software Prerequisites

The trust tool set runs on Java (runtime environment 6 or later). Development has been done on Windows XP SP3 and testing on Windows, Solaris and Ubuntu though it does not use platform specific code and should work on any platform running the Java VM.

The Trust PDP requires the following external components:

Component	Address	Tested Jar Version
Sun XACML	http://sunxacml.sourceforge.net/,	build from unstable in SVN, 08-10-2009
Sun XACML-support	http://sunxacml.sourceforge.net/	build from unstable in SVN, 30-11-2009
Apache logging	http://logging.apache.org/log4j/1.2/index.html	log4j-1.2.15.jar
JDOM Java XML tools	http://www.jdom.org/	jdom.jar (version 1.0)

The centrality (RTM) trust service interface requires the following external component:

Component	Address	Tested Jar Version
PostgreSQL JDBC drivers	http://jdbc.postgresql.org/download.html	postgresql-8.3-604-jdbc3.jar

The Polipo (CTM) trust service and its interface require network connectivity (the service is inherently decentralized) and the following components:

Component	Address	Tested Jar Version
Apache Tomcat	http://tomcat.apache.org/	version 6.0.26

The service quality (OCTTM) trust service and its interface require the following components:

Component	Address	Tested Jar Version

XML Beans	http://xmlbeans.apache.org/	xmlbeans-2.3.0.jar
Test result document parser	Component T3-OCT software pack (TAS3 portal - public release to follow).	rolecastTestResultsXMLTypes.jar

The Unit testing requires the following external components:

Component	Address	Tested Jar Version
Unit testing	http://www.junit.org/	junit-4.4.jar
XML Unit test support	http://xmlunit.sourceforge.net/	xmlunit-1.2.jar

3.2 Installation and Configuration Instructions

The suggested installation and testing order is as follows: ensure the Java runtime environment (tested version: v6) is up and running. Next get the Trust PDP core running. After this is working and passes the basic tests install and configure the required trust services. If the configuration is running correctly through the Java interface reuse the configuration files as setup for the web service interface.

3.2.1 Installing the Trust PDP

Extract trustpdp.jar, and libraries (trustpdp_lib*.jar) from the component zip file. Running "java –jar trustpdp.jar" should now work and give a short help message. (When using the source code from the SVN the starting class to use is `eu.tas3.trustpdp.TrustPDPBasic`.)

Create a configuration file (e.g. config.xml) which points at to two files containing the trust service setup (e.g. trustservices.xml) and trust policies to use (e.g. trustpolicies.xml) respectively. Optionally assign a name to this configuration using the PolicyIdentifier element.

```
<?xml version="1.0" encoding="UTF-8"?>
<TrustPDPConfiguration>
  <PolicyIdentifier>StandardConfig</PolicyIdentifier>
  <PolicyConfigFile>trustpolicies.xml</PolicyConfigFile>
  <TrustServiceConfigFile>trustservices.xml</TrustServiceConfigFile>
</TrustPDPConfiguration>
```

Select the trust services to use and configure these in the trust service configuration file (trustservices.xml).

```
<?xml version="1.0" encoding="UTF-8"?>
<config defaultTrustServiceFactory="TUE-TrustServices">
  <trustserviceFactory name="TUE-TrustServices">
  <list>
```

```
      <trustService class="eu.tas3.trustpdp.rtm.TrustService">
            <Registry>security1.win.tue.nl</Registry>
      </trustService>
      <trustService class="eu.tas3.trustpdp.ctm.TrustService"/>
      <trustService class="eu.tas3.trustpdp.octtm.TrustService"/>
      <trustService class="eu.tas3.trustpdp.genericctm.TrustService"/>
      <trustService class="eu.tas3.trustpdp.simtm.TrustService">
        <SimulatedFunction>
          <Name>sim:halftrusted</Name>
          <DefaultValue>0.5</DefaultValue>
          <Score Ratee="Alice">1.0</Score>
          <Score Ratee="Mallory">0.0</Score>
        </SimulatedFunction>
      </trustService>  </list>
  </trustserviceFactory>
</config>
```

For a first test just configure the `Generic Trust Management' trust service and the `trust everybody policy' (allow_all_policy.xml), send any valid request to the server (see Section 3.3) and a PERMIT response should be returned. Next configure additional trust services as described below, add them and test them with the basic policies/request provided (see Section 3.3). Any configuration information needed by a trust service is included in the element, such as the RMI-registry to use for locating the RTM service gateway in the example configuration above (see Section 3.2.2 for more details on the RTM gateway).

Set up the policies that are to be used by the trust service. The trust policy configuration file specifies policy finder modules. Both static policies and references to other policies are supported. The configuration below will load policies from allow_all_policy.xml and will also check this file for references to other policies. Other policy finders may also be used.

```
<?xml version="1.0" encoding="UTF-8"?>
<config defaultPDP="TrustPDP" defaultAttributeFactory="attr"
defaultCombiningAlgFactory="comb" defaultFunctionFactory="func">
<pdp name="TrustPDP">
  <policyFinderModule
class="com.sun.xacml.support.finder.StaticPolicyFinderModule">
      <list>
      <string>allow_all_policy.xml</string>
      </list>
  </policyFinderModule>
  <policyFinderModule
class="com.sun.xacml.support.finder.StaticRefPolicyFinderModule">
       <list>
      <string>allow_all_policy.xml</string>
      </list>
  </policyFinderModule>
  <policyFinderModule
class="com.sun.xacml.support.finder.URLPolicyFinderModule" />
  <attributeFinderModule
class="com.sun.xacml.finder.impl.CurrentEnvModule" />
  <attributeFinderModule
class="com.sun.xacml.finder.impl.SelectorModule" />
</pdp>
  <attributeFactory name="attr" useStandardDatatypes="true" />
```

```
  <combiningAlgFactory name="comb" useStandardAlgorithms="true" />
  <functionFactory name="func" useStandardFunctions="true" />
</config>
```

The trust policy file gives a XACML policy set where any trust functions implemented by the configured trust services can be used in the conditions.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="TSP:Rep:requester"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">
  <Description>
    Allow read if sufficient reputation.
  </Description>
  <Target>
    <Subjects> <AnySubject/> </Subjects>
    <Resources> <Resource>
      <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">myfile</AttributeVal
ue>
      <ResourceAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
id"/>
      </ResourceMatch>
    </Resource> </Resources>
    <Actions> <AnyAction/> </Actions>
  </Target>
  <Rule RuleId="nurse:role:reptrustmetric" Effect="Permit">
    <Target> <Subjects> <AnySubject/> </Subjects>
    <Resources> <AnyResource/> </Resources>
    <Actions> <Action>
      <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue
>
        <ActionAttributeDesignator
         DataType="http://www.w3.org/2001/XMLSchema#string"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
    </Action> </Actions>
    </Target>
    <Condition
FunctionId="http://localhost:8080/trustpdp/names/function#reputation">
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
          <SubjectAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
   </Apply>
   <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">pagerank</AttributeV
alue>
```

```
    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#double">0.8</AttributeValue>
 </Condition>
 </Rule>
 <Rule RuleId="DenyAllOthers" Effect="Deny"/>
</Policy>
```

The example policy above has a single rule that allows read access to myfile to anybody with a reputation of at least 0.8 according to the pagerank centrality measure (and a default rule denying all others). Note that to get the identity of the user for which the reputation needs to be computed (i.e. the requester subject) the Apply function is used to take one of the arguments from the request.

3.2.2 Supporting a RTM Service

See [D5.2] for information on deploying a RTM trust service. For the Trust PDP interface to the RTM service no additional setup is needed. Simply ensure the required libraries are installed and add the server object to the trust service configuration file (the trustservices.xml file in the example above) (`<trustService class="eu.tas3.trustpdp.rtm.TrustService">`). Optionally set the address and port using a <Server> element with address and port settings. Ensure that the address and port are reachable. (Without this optional setting the standard RTM service running at the KARL site is used. Access to this service may require a firewall exception at that site granting access from the location running the Trust PDP (or the gateway to the RTM service.))

The connection to the RTM Service can be done through a gateway. To do this run startRTM.bat/startRTM.sh which runs eu.tas3.trustpdp.centralityTM.TrustService with as argument a configuration file (the format is this file is the same as that of trustservices.xml; actually this document can be used as argument). Several setting can be configured for the RTM service running as a gateway (given with type and default value):

```
      ServicePort  - Port for the gateway to listen to (integer, 0)
      Registry     - address of the registry (url, security1.win.tue.nl)
      RegistryPort - port used by the registry (integer, 1099)
      ServiceName  - name of the service at the registry (string,
TrustService-RTM)
```

The ServicePort is only relevant when starting the RTM TrustService as a gateway. The others should be set to the same value for both the gateway and the Trust PDP which connects to this gateway.

Note that other trust services can also be contacted through a gateway in the same way. However, so far no need for this has been encountered.

A RTM unit test is available to check correct configuration and reachability of the service (`eu.tas3.trustpdp.centralityrtm.RTMTestCase`).

3.2.3 Supporting a CTM Service

Add the server object to the trust service configuration file (trustservices.xml) (`<trustService class="eu.tas3.trustpdp.ctm.TrustService"/>`).

A CTM unit test is available to check correct installation and reachability of the service (`eu.tas3.trustpdp.ctm.CTMTestCase`).

3.2.4 Supporting an OCTTM Service

Add the server object to the trust service configuration file (trustservices.xml) (`<trustService class="eu.tas3.trustpdp.octtm.TrustService"/>`).

A OCTTM unit test is available to check correct configuration of the service (`eu.tas3.trustpdp.octtm.OCCTMTestCase`). To validate that the service is correctly receiving test messages from the audit bus an OCT test needs to be triggered. The logs will show the result of the test. Scripts (`octtrack.sh/bat/php`) are available to extract these results from the log and generate an html representation or a live monitoring page.

By evaluating trust function "octtm:Reset" offered by the OCTTM service, for example by requesting it through the Trust Data Access Service Provider (TDASP, see also Section 4.1) interface, the OCT test scoring data base can be reset, allowing testing to start from a fresh configuration. (This specific testing functionality should likely be disabled in a production environment. See Section 4.2.1 for details on changing functions supported by a trust metric. It could be replaced by a simulated function, see Section 5.3.1.)

3.2.5 Deploying the Trust PDP as a webservice:

This uses the standalone authorization server developed by the University of Kent [PERMIS] standalone server adapted to call the Trust PDP rather than the SUN XACML PDP. Extract the standalone server to a directory. Configure the desired port, the configuration file containing the trust services and the configuration containing the trust policies in permis.xml.

```
<PERMISStandaloneConfiguration>
  <TCPConfiguration>
     <ServerPort>1104</ServerPort>
     <ThreadCount>10</ThreadCount>
     <Protocol>http</Protocol>
     <RequireClientAuthentication>false</RequireClientAuthentication>
  </TCPConfiguration>

  <!-- trust PDP configuration: -->
  <TrustPDPConfiguration>
     <PolicyIdentifier>ReputationConfig</PolicyIdentifier>
     <PolicyConfigFile>trustpolicies.xml</PolicyConfigFile>
     <TrustServiceConfigFile>trustservices.xml</TrustServiceConfigFile>
  </TrustPDPConfiguration>
</PERMISStandaloneConfiguration>
```

Ensure the XML libraries in lib/endorsed are endorsed. The easiest method to achieve this is to copy them to `$JAVA_HOME/jre/lib/endorsed`.

This will allow use of the always trusted, CTM and OCTTM services. For the RTM service an RMI-gateway should be used to avoid conflicts between the Trust PDP webservice interface and the trust service interface. Use of the wrappers is selected through the configuration of the RTM trust service (see Sections 3.2.1 and 3.2.2).

Run standalone.bat (windows) or standalone.sh (unix) to start the server.

3.3 Running the Tests

The following unit tests are available:

To test the (connection to the) RTM service the class `eu.tas3.trustpdp.rtm.TestCase` offers two test; testEvaluate() executes a basic reputation query, testEvaluateWithAccreditedRater() runs a nested query where only the feedback from trusted raters is to be taken into account.

To test the (connection to the) CTM service the class `eu.tas3.trustpdp.ctm.TestCase` offers test testEvaluateSetSubjectVar() which queries a fixed credential. Several other unit test routines check internal functionality of the CTM interface.

To test the OCTTM service can correct connection to and registration at the TAS³ Auditbus, the class `eu.tas3.trustpdp.octtm.TestCase` offers test testRegister() which will set up a listener, wait for a message and print information on the received message. (To complete this test an OCT test case message should be posted to the auditbus.)

Besides the unit test several simple test policies are provided in resources/policies to test the functioning of the different trust services. The policy allow_all_policy.xml allows all valid requests and can be used to check that the basic TrustPDP setup is working, independent of the trust services. A basic reputation policy ReputationPolicy.xml will test that the corresponding service is running. In resources/gem a basic credential policy CredentialPolicy.xml and agent credential policies (capturing the credentials issued by those agents) is provided. To run these tests; update the policy configuration files to select the policy and the trust service configuration file to include the service to be tested. A `Permit' response indicates a successful test. A 'not applicable' response (possibly with warnings from the policy finder) indicates the policies could not be read. A ConnectionManager 'cannot create connection' error would indicate an unreachable trust service while other errors indicate an incorrect configuration/setup of the trust tool set itself.

Potential issues;

1. Spaces in the pathnames of the config/policy files can give problems, avoiding these is advised.

2. The CTM service is inherently distributed; it will, for all but the simplest cases, need network connectivity to query CTM services of other agents.

4 How to Use the Software

This section describes how to use the trust tool set in evaluating trust as part of the TAS³ authorization process [D7.1]. It addresses configuring the system for a given scenario, creating policies, and calling the Trust PDP to check trust requirements.

4.1 Tutorial

We use an example in the setting of the employability integration test scenario described in [D9.1]. This scenario has been mapped to a business process in [D3.3] (see Figure 4.1 `Nottingham process' for the process on which we base this example). The key steps in this example are (see [D9.1] for a complete scenario description):

1. Learner Alice seeking placement registers with placement administrator Bob.

2. Alice is presented suitable programmes and selects one.

3. Alice provides information needed for the selected program; her CV. Alice sets her requirement on access to her data by selecting policies. Different types of requirements are used, each expressed in the policy. Alice's trust policies require service providers with a good reputation amongst students.

4. Service negotiation is used to discover suitable services providers.

5. Alice selects a service provider. The provider is contacted and offers a choice of services from which Alice chooses the `Intern position matching service'. The service provider retrieves Alice's CV and determines and returns matching vacancies.

6. The results are presented to Alice, Alice gives feedback on the service provided by the provider.

Further steps in the application process are out of the scope of this example process.

In the first step Alice is authenticated details on the authentication process are provided in [D7.1]. Here we will simply assume that identity `Alice' has been determined and certified to be an eligible student. In the third step Alice provides her personal information along with her requirements. To implement the protection Alice's requirements need to be expressed as policies and loaded into the Trust PDP:

A. Create a policy file for Alice's requirements:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="TSP:Rep:requester"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">
  <Description>
    Allow read if sufficient reputation amongst student.
  </Description>
  <Target>
    <Subjects> <AnySubject/> </Subjects>
```

```
    <Resources> <Resource>
      <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">CV-
Alice</AttributeValue>
      <ResourceAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
id"/>
      </ResourceMatch>
    </Resource> </Resources>
    <Actions> <AnyAction/> </Actions>
  </Target>
  <Rule RuleId="nurse:role:reptrustmetric" Effect="Permit">
    <Target> <Subjects> <AnySubject/> </Subjects>
    <Resources> <AnyResource/> </Resources>
    <Actions> <Action>
      <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue
>
        <ActionAttributeDesignator
         DataType="http://www.w3.org/2001/XMLSchema#string"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
        </ActionMatch>
    </Action> </Actions>
    </Target>
    <Condition
FunctionId="http://localhost:8080/trustpdp/names/function#reputation-
by-accredited-rater">
      <!-- Trusted raters: Nothingham.Students -->
      <Apply
FunctionId="http://example.org:8080/trustpdp/names/function#credential-
subject-list">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Nottingham</Attribut
eValue>
        <AttributeValue
             DataType="http://www.w3.org/2001/XMLSchema#string">Student
                                                    </AttributeValue>
      </Apply>
      <!-- The requester (subject id) -->
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
one-and-only">
        <SubjectAttributeDesignator
                    DataType="http://www.w3.org/2001/XMLSchema#string"
       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
      </Apply>
      <!-- The reputation metric to use -->
      <AttributeValue
 DataType="http://www.w3.org/2001/XMLSchema#string">pagerank</AttributeV
                                                              alue>
      <!-- The required score on this metric -->
      <AttributeValue
 DataType="http://www.w3.org/2001/XMLSchema#double">0.8</AttributeValue>
    </Condition>
  </Rule>
  <Rule RuleId="DenyAllOthers" Effect="Deny"/>
```

```
</Policy>
```

The policy uses the reputation by accredited (i.e. trusted) raters function. This function takes a list of trusted entities and takes only their feedback into account when computing the reputation. This list of entities in turn is computed by the credential subject list function which gives the entities with a given credential (i.e. students at Nottingham in this example). The second argument to the reputation function is the entity whose reputation needs to be calculate (the requester, i.e. subject of the request in this example). The third argument gives the reputation centrality metric to use while the last gives the score that needs to be achieved on this metric. See section 5.3.1 for the currently available trust functions and how to provide additional trust functions.

Safe the policy in a file, e.g. AlicePolicy.xml

B. Update the Trust PDP configuration

Create a PDP configuration file as in Section 3.2.1. The policy of Alice uses the RTM and CTM services. Ensure these are listed in the trustservice configuration file (trustservices.xml if the example configuration of Section 3.2.1 is used). Add the policy of Alice to the list of policies in trust policy configuration file (trustpolicies.xml).

C. (Optional) Test the PDP

Create a sample request in request.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute
           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>ServiceX</AttributeValue>
    </Attribute>
  <Resource>
    <Attribute
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>CV-Alice</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute
             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                    DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

and check the response of the Trust PDP

```
$ java -jar TrustPDP.jar -config config.xml request.xml
```

A permit or deny response should be issued depending on the reputation of *ServiceX*.

D. Start the Trust PDP

See the instructions in section 3 on how to deploy the Trust PDP as either a Java program or a web service.

E. Configure the PEP to use the PDP (out of the scope of this document).

In step 4 suitable services providers are discovered. For potential providers we check whether they satisfy Alice's requirements using the Trust PDP. We can make the distinction between trusted and untrusted providers but if multiple providers are trusted Alice may wish to use the one with the highest reputation or, if no suitable providers are found she may wish to lower the required reputation score. This is where *Trust Data Access Service Provider (TDASP)* comes in; its functionality is similar to the Trust PDP except that it also returns trust ranking scores if the entity is trusted. (The TDASP functionality has been built into the Trust PDP thus running it only requires configuring a Trust PDP as given below.)

In this example Alice's requirement consists of a single trust metric and thus the metric we would like to use is clear. However, in general a policy can contain many different metrics which are combined in different ways. Thus in general the policy does not tell us which trust metric to use for ranking requiring us to set this ranking separately. We can do this statically as follows: an extra configuration element `<Ranking>` with sub elements `<Service>` and `<Metric>` in the main configuration file are used for this.

```
<?xml version="1.0" encoding="UTF-8"?>
<TrustPDPConfiguration>
  <PolicyIdentifier>StandardConfig</PolicyIdentifier>
  <PolicyConfigFile>trustpolicies.xml</PolicyConfigFile>
  <TrustServiceConfigFile>trustservices.xml</TrustServiceConfigFile>
  <Ranking>
    <Service>eu.tas3.trustpdp.octtm.TrustService</Service>
    <Metric>oct:TestCount</Metric>
  </Ranking>
</TrustPDPConfiguration>
```

In the example above the score on OCT metric `TestCount' is used for ranking services. Additional configuration information may also be added to the Ranking element as needed.

Typically the most useful static ranking for the TDASP is created by placing the most important criterion which is useful for ranking (e.g. credential based trust criteria normally have only two trust levels which allows only very limited ranking) as the ranking metric rather than as a trust requirement in the policy.

In addition to the possibility of statically specifying the ranking to return in the configuration file of the Trust PDP it is also possible to dynamically specify metrics for which one wants a ranking in the request sent to the TDASP/Trust PDP. This is done by setting an environment attribute in the request:

```
<Environment>
```

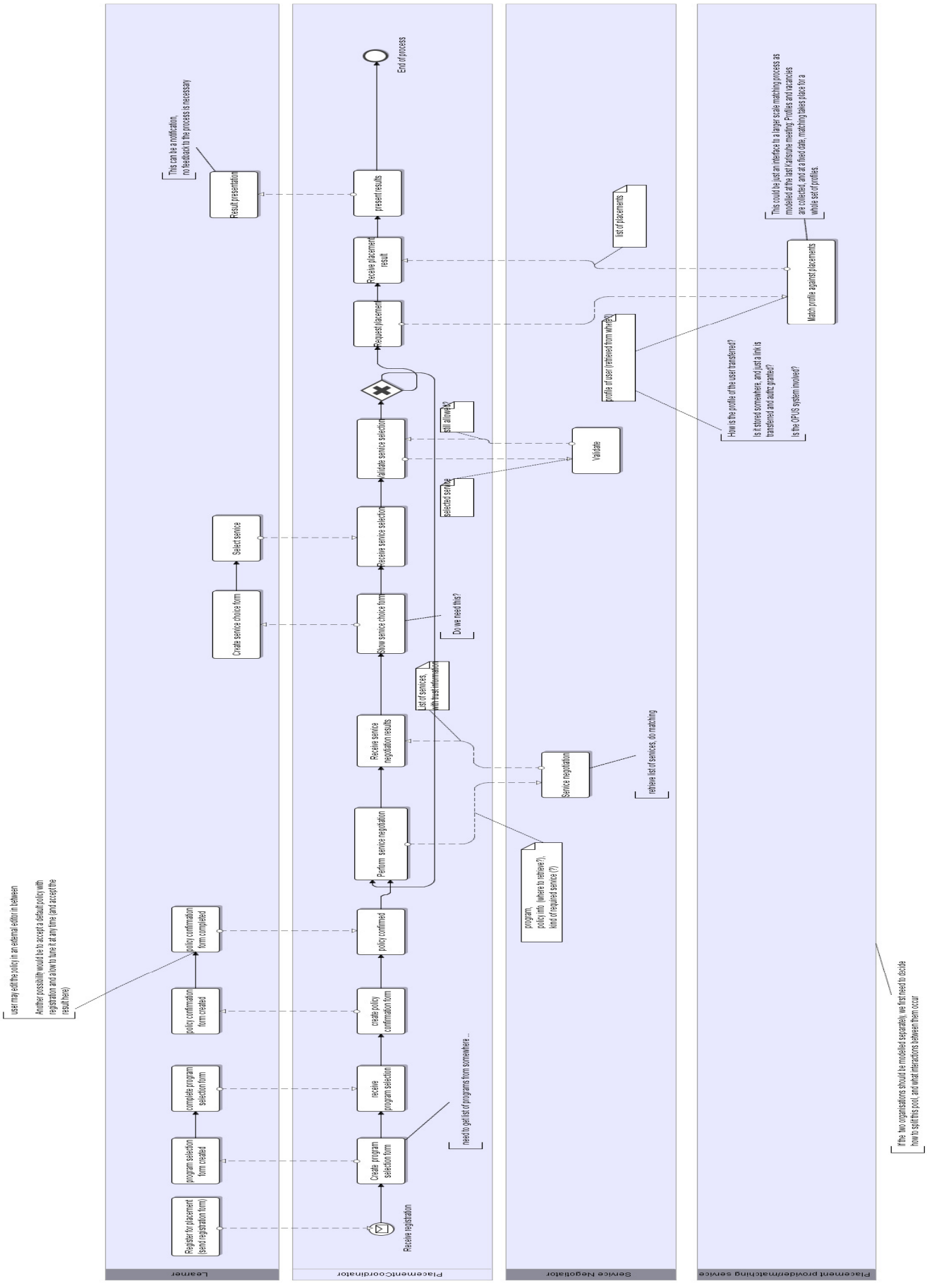
```
<Attribute AttributeId = "urn:tas3:trust:input:clt1:ranking"
        DataType = "http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>x:y#config</AttributeValue>
</Attribute>
```

Where `x:y` is the name of the metric with `x` the sub name space of a trust service and `y` the name of a trust metric offered by this trust service. See Section 4.2.1 for details on naming conventions and Section 5.3.1 for available functions. The name is optionally followed by # and a string `config` providing configuration information (such as the centrality measure to use to calculate reputations or the issuer and role for a credential metric) for the trust metric.

The limitation for dynamically specified rankings, however, is that the trust metrics used to rank must use a fixed format for its arguments (See `Metrics for Trust Data Access Service Provider (TDASP)' in Section 4.2.1).

Multiple rankings can be obtained by specifying multiple attributes. If the trust evaluation results in a PERMIT, all rankings (the static one if set and each of the dynamic ones in the request) are returned as sub-status codes in the XACML response.In step 5 Alice actually uses the service. The Trust PDP will again be used when the service provider tries to access her CV. Note that, even though the service was selected from trusted services success of this call is not guaranteed. For example; the reputation of the provider may since have decreased due to recent negative feedback. In this case the business process would have to adapt to the fact that the selected service is no longer suitable. (Further details are out of scope of this document.)

In step 6 Alice has the opportunity to provide feedback on the service provider, influencing its reputation. See [D5.2] for details on the reputation service and the corresponding feedback service.


Learner
PlacementCoordinator
Service Negotiator
Placement provider/matching service
Register for placement (send registration form)
program selection form created
complete program selection form
policy confirmation form created
policy confirmation form completed
user may edit the policy in an external editor in between
Another possibility would be to accept a default policy with registration and allow to tune it at any time (and accept the result here)
Receive registration
Create program selection form
need to get list of programs from somewhere ...
receive program selection
create policy confirmation form
policy confirmed
Perform service negotiation
program, policy info (where to retrieve?), kind of required service (?)
Service negotiation
retrieve list of services, do matching
Receive service negotiation results
list of services, with trust information
Do we need this?
Allow service choice form
Create service choice form
Select service
Receive service selection
selected service
still allowed?
Validate service selection
Validate
profile of user (retrieved from where?)
How is the profile of the user transferred?
Is it stored somewhere, and just a link is transferred and authz granted?
Is the OPUS system involved?
Request placement
list of placements
Receive placement result
present results
End of process
Result presentation
This can be a notification, no feedback to the process is necessary
This could be just an interface to a larger scale matching process as modelled at the task/subscher meeting: Profiles and vacancies are collected, and at a fitted date, matching takes place for a whole set of profiles.
Match profile against placements
If the two organisations should be modelled separately, we first need to decide how to split this pool, and what interactions between them occur

4.2 Developer Tutorial

In this section we illustrate the setup and use of the Trust tool set from a developers point of view - it is meant to support those developing trust services and trust functions for use with the trust PDP and as such gives a higher level of technical details than is needed for deploying the software.

4.2.1 Adding new Trust Metrics to existing services

Trust metrics (see [D5.1]) are implemented as specific types of XACML user defined functions, namely `eu.tas3.trustpdp.trustservice.TrustMetric` objects. Trust policies are then standard XACML policies using these user defined functions to refer to trust metrics (see example in section 4.1 step A). Subclasses of the TrustMetric class, such as `eu.tas3.trustpdp.octtm.TrustMetric`, define trust metrics offered by a specific trust service (the OCT trust management service in this example).

Trust services are used to evaluate the trust metrics. A trust service may offer a single metric which is fully configured by its arguments or offer several trust metrics with varying amounts of configurability covering different types of trust evaluation offered by the service. Note that the Trust Data Access Service Provider only supports metrics with the argument format given at the end of this section.

Adding a new trust metric to an already supported trust service requires extending the TrustMetric class used for that service according to the following steps (assuming the standard template was followed in creating the trust service):

- A name and sequence number (function ID) are chosen for the metric. The name may not contain the characters ':' and '#'. Meaningful XML sequences may also be problematic and should thus be avoided. (Using alphanumerical, underscore and dash characters only is advised.). Defining a constant for the function ID is advised and the metric's name must be added in the correct location in the array FunctionNames used in the class initialization.

```
public static final int ID_MTS_MY_TRUST_METRIC = 2;
static { ...
         String FunctionNames[] = { ..., ..., "MyTrustMetric" };
         ...
}
```

  Here function "MyTrustMetric" is the third (counting starts at 0) function supported by the trust service MyTrustService (MTS).

- Next, the types of the arguments of the metric are specified by adding them in the multidimensional array `ParameterTypes'.

```
String ParameterTypes [][] =
{ ..., ...,
  // MyTrustMetric: String ratee, Double my configuration info
  { StringAttribute.identifier, DoubleAttribute.identifier },
};
```

  The example function `MyTrustMetric' takes two arguments; the principle to rate and a real number used for configuration.

- An argument can either be of a basic type (Strings, Doubles, etc.) or can be a collection (bag) of elements of the same basic type. The multidimensional array areBags, which has the same size as ParameterTypes, specifies for each argument whether it is a bag.

```
boolean[][] areBags = { ..., ..., { false, false } };
```

  Both arguments to `MyTrustMetric' (the ratee and the configuration number) are single items.

This completes the type specification of the trust function. Next, its behaviour, i.e. the outcome of the function needs to be specified. The default behaviour (described in more detail below) may already be sufficient but can be customized as follows:

- *Customized queries:* A trust metric is evaluated for a given ratee by querying the appropriate trust Service. The query, which captures the metric and its configuration information in the form of a `String`, is created by the method `buildMetric`. The default value for the query is the name of the metric and an optional argument separated by #. Appropriate translation code should be added to `buildMetric` if this default is not sufficient; e.g. because the trust service uses a different format for its queries (see e.g. `eu.tas3.trustpdp.rtm.TrustMetric` for examples of non-trivial translations from metrics and arguments to queries).

```
public String buildMetric( AttributeValue[] attributes )
{
   switch ( getFunctionId() )
   { case ID_MTS_MY_TRUST METRIC:
         return "MTF<"
               + ((DoubleAttribute) attributes[1]).getValue()
               + ">";
     default: return super.buildMetric( attributes );
       // super gives: getFunctionName() + (attribute[1] is a String)?
       //   "#" + ((StringAttribute) attributes[1]).getValue() : "";
   }
 }
```

  Function MyTrustMetric is implemented by having the trust service evaluate query "MTF<*nr*>". Note that the first argument to MTF, the ratee, does not appear in the query; it is passed as a separate argument to the trust service.

- *Alternative processing:* The default processing evaluates a trust metric by first evaluating its inputs, then formulating a query with `buildMetric` as described above and sending this query to the trust service. The score given by the service is then returned in the form of an XACML `EvaluationResult`. Normally this default processing approach should suffice. However, this can also be customized by adapting the method `evaluate`, adding a case for the metric requiring an alternate processing approach:

```
public EvaluationResult evaluate( List inputs, EvaluationCtx context )
{
   switch ( getFunctionId() )
   { case ID_MTS_MY_TRUST_METRIC: {
         ... special processing to create query ...
         String query = ...
         String ratee = ...
```

```
            float result = connector.evaluate( query, ratee, false );
            ... post processing of result ...
            return new EvaluationResult( new DoubleAttribute(result) ); }
        default: return super.evaluate( ID_TM_DEFAULT, inputs, context );
    }
}
```

In this way the evaluation for MyTrustMetric can be customized as needed. Note that this does mean that all processing steps must be specified for this metric. (Function *ID_TM_DEFAULT* in `eu.tas3.trustpdp.trustservice.TrustMetric` can be used as a starting point; it provides the default processing described above.)

This completes the addition of a new trust metric to an already supported trust service. In Section 4.2.2 we treat how to add support for an entirely new trust service.

TrustMetrics versus TrustFunctions

Trust metrics return a level of trust which is then compared to a required level in a policy (Trust PDP) or returned (TDASP). The XACML user defined functions which are used to implement the trust metrics may use different return types. Some trust evaluation results in a yes/no type of answer rather than a level. Though this can, of course, be coded as a system with two levels, in policies it is more natural to interpret such an answer as the result directly rather than comparing it to a level (saying `trusted' rather than 'trusted is at least level yes'). In the Trust PDP this is supported by using `eu.tas3.trustpdp.trustservice.TrustFunction` objects which return a Boolean (rather than `eu.tas3.trustpdp.trustservice.TrustMetric` objects with return a level).

Metrics for Trust Data Access Service Provider (TDASP)

Trust metrics which are to be evaluated in dynamic requests to the TDASP (See Section 4.1) should take as its first argument a `String` which will hold the ratee and may optionally use a second `String` argument which encodes any arguments to the function.

4.2.2 Creating new Trust Services

An `eu.tas3.trustPDP.TrustService` object (TrustService) is the local representation of a trust service offered somewhere within the TAS³ trust network. The following template can be used to add a new TrustService to those supported by the Trust PDP. (The existing services such as RTM, OCTTM and CTM illustrate this template.)

The template uses three classes: A TrustMetric class which expresses the different supported trust metrics (as in Section 4.2.1 above), a Connector class which provides the actual connection from the local representation to the real trust service, and a wrapper TrustService class which initializes these classes and integrates them into the TrustPDP's XACML based request evaluation structure.

- A (unique) name ***x*** is selected for the trust service. The default name for the new TrustService class is `eu.tas3.trustPDP.`***x***`tm.TrustService`. In addition to the TrustService interface, this class should also implement the interface `com.sun.xacml.cond.cluster.FunctionCluster` (FunctionCluster).

- Several existing trust services implement the `eu.tas3.trustPDP.RankingService` (RankingService) interface extension of the TrustService interface. This interface adds mostly obsolete functionality to the TrustService interface; it is used by the TDASP to return rankings based on trust metrics specified in its configuration file. As a more flexible dynamic querying mechanism for obtaining rankings has been added, as described in Section 4.1, the staticly configured method of accessing trust data functionality is normally not needed.

- The TrustService should offer a constructor which takes a single argument of type `org.w3c.dom.Element` (Element) which is used to pass configuration information specified in the TrustPDP configuration file (see example trustservice.xml in Section 3.2.1) to the TrustService. A no argument constructor may be used instead if no configuration information is needed.

  One of the jobs of the TrustService constructor is setting up a `eu.tas3.trustPDP.trustservice.Connector` (Connector) object (see also discussion on creating a Connector below). In the template we assume a variable `connector' is used to link to the created connector. The existing classes do not all follow this template - some use a singleton connector (e.g. OCTTM) so do not need to store the link while others use a wrapper around the connector (e.g. RTM service) which is then used instead.

- The second method which must be present is `getSupportedFunctions` from the interface FunctionCluster. The default for this method is to create a function for each metric supported by the `eu.tas3.trustPDP.`***x***`tm.TrustMetric` (TrustMetric) class:

```
Set set = new HashSet();
Iterator it = TrustMetric.getSupportedIdentifiers().iterator();
while (it.hasNext())
      set.add( new TrustMetric((String)(it.next()), connector) );
return set;
```

  (See also the discussion on TrustMetric and Connector creation below.) Functions from `eu.tas3.trustPDP.`***x***`tm.TrustFunction` can be added in the same way should they be present.
- Finally the `evaluate` method specified in the TrustService interface must be present. Simply passing the already correctly formatted request directly to the connector should suffice. (Often this will not even be used; the TrustMetrics will call the connector directly.)

```
return connector.evaluate( metric, requester, validateResponse );
```

This completes the TrustService class. Next steps are the creation of the TrustMetric and Connector classes. Starting with the former, we describe the creation of a basic skeleton to which metrics can then be added as described in the previous section.

An `eu.tas3.trustpdp.trustservice.TrustMetricDomain` (TrustMetricDomain) is the specification of the trust functions belonging to a single trust service. It uses its own unique sub-namespace (by default stored in constant *SERVICE_SUBNS)* within *eu.tas3.trustpdp.trustservice.TrustMetric.RESEARCH_NS*, the trust metric name space. This ensures that no conflicts between metric names are possible between trust

services. This sub-namespace is created, initialized and registered in the static class initialization of the TrustMetric:

```
static {
      String[] functionNames = { };
      String[][] parameterTypes = {};
      boolean[][] areBags = {};
      addSubNameSpace( new TrustMetricDomain( SERVICE_SUBNS,
                  functionNames, parameterTypes, areBags ) );
}
```

This static class initialization code defines an empty set of supported metrics and registers them in their own sub-namespace (again, See 4.2.1 for adding actual metrics). Method `addSubNameSpace` that does the registration is offered by the parent class `eu.tas3.trustpdp.trustservice.TrustMetric`.

The default constructors simply pass the arguments to the Parent. Instance based initialization can be added here should any be necessary. The method `getSupportedIdentifiers`, implementing part of the `FunctionCluster` interface, obtains the registered functions for this sub-namespace from the Parent. This method must be present but it should be possible to use the code in the template without changes.

The method `buildMetric` needs only be overwritten, following the approach described in Section 4.2.1, if one of the trust metrics does not use the default method. The method evaluate described the behaviour of the functions and must always be present. Normally using the default processing offered by function *ID_TM_DEFAULT* of Parent `eu.tas3.trustpdp.trustservice.TrustMetric` suffices:

```
return super.evaluate( ID_TM_DEFAULT, inputs, context );
```

This completes the TrustMetric skeleton.

A Connector is medium specific; it is responsible for making the actual connection from this local stub to the actual physical trust service. This could be by local call, a web service call, html(s), rmi, a database connector, the TAS³ auditbus, etc. (Note that the integrity and confidentiality of queries and resulting scores should be evaluated for a chosen solution.) The connector sends out the ratee, the metric to be evaluated for this ratee, optionally the values for embedded metrics, and obtains a resulting score. As the implementation will depend on the medium used, the template does not specify this further. The connectors employed by the different trust services provide code for several media.

This completes the integration of a new TrustMetric in the TrustPDP.

5 Architecture

5.1 Position in the TAS³ Architecture

Figure 5.1 shows the TAS³ authorization framework [D2.1, D7.1]. It show the core idea of using a master PDP to combine policy decisions from different types of policies; be it XACML [XACML] policies, PERMIS [PERMIS] policies, Trust policies, etc. The interaction between the web service and the Master PDP is described in [D7.1]. Here we focus on the Trust management sub architecture (the `Trust' box of the diagram in Figure 5.1) and its interaction with the Master PDP. The Master PDP and each of the sub PDPs share the same interface: They evaluate a request formulated as a XACML request context and respond with a XACML response context. This allows direct use of the sub PDPs if only one such PDP is needed. Again, see [D7.1] for the details on how the web service uses a PDP.

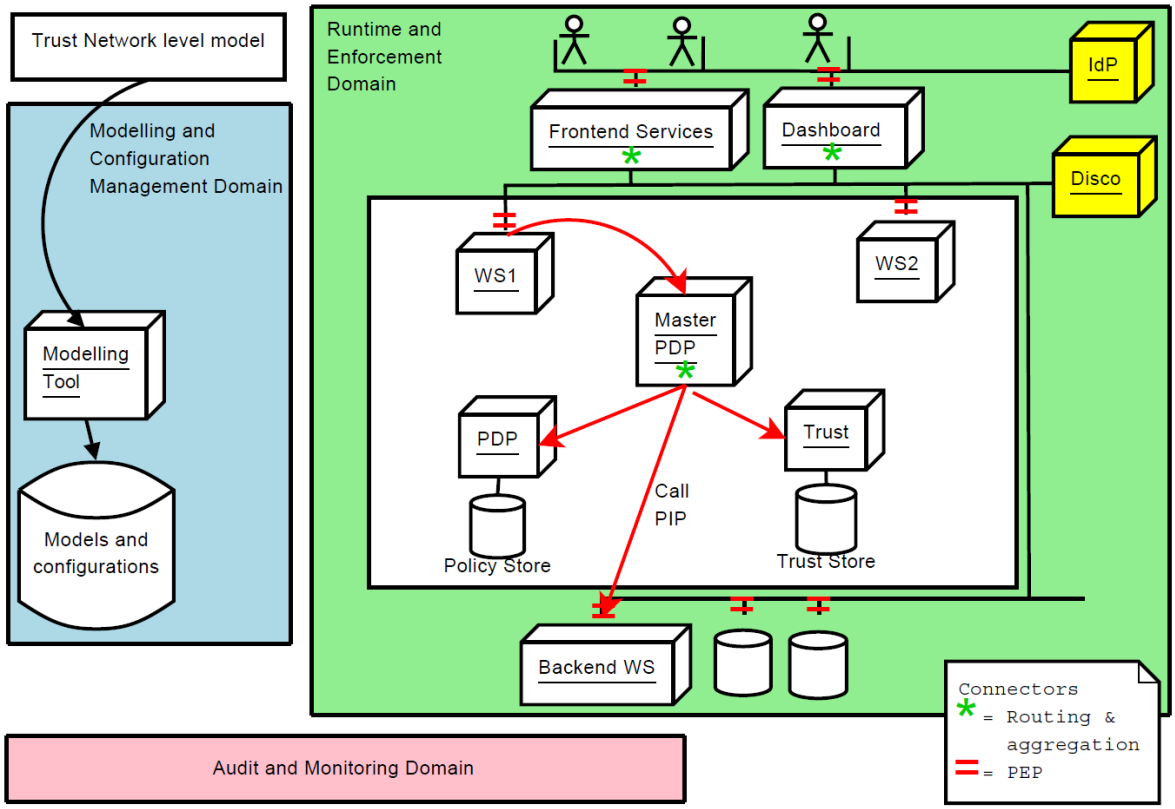


Figure 5.1 Authorization Process [D2.1]

As mentioned above the Trust PDP evaluates standard XACML requests. The request describes a subject, a resource and an action on the resource. Optionally the request can also describe the environment. The Trust PDP evaluates whether the subject is trusted to perform this action on the resource based on the available trust policies. (Policies are installed in the Trust PDP, however, a Trust PDP can be created passing a `dynamic policy' which is added to the policies given by the (static) configuration

information. This allows the master PDP to spawn an appropriate Trust PDP when it encounters a sticky policy [D7.1] whose content is a trust policy.

As mentioned in Section 2.3, Section 4.2 and examples above, the trust policy language [D5.1] is implemented in the XACML language enhanced with *trust functions*. Trust functions capture trust requirements, typically of the requester. For example a trust function may express that the requester must have a trust attribute certified by the hospital or a reputation score of at least 0.8 according to a given reputation calculation schema. The following sections describe how the Trust PDP uses function wrappers around calls to trust services to implement the trust functions.

5.2 Position in the Trust Architecture

As described above, the main interface of the trust policy architecture is the Trust Policy Decision Point (Trust PDP) which is called by the master PDP in the TAS³ authorization architecture. The policies that are evaluated by the trust PDP capture different types of trust criteria as trust can depend on different types of information. Trust services evaluate trust information and translate these into levels of trust which can then be compared to the required level set in the trust criteria. Figure 5.2 illustrates this: The trust policy that is set in the trust PDP requires either a high score on the reputation metric (the actual metric and required score are not displayed) or a credential (the issuer and trust attribute are not displayed). When evaluating a request the trust PDP needs to checks the conditions in the rules. The trust functions that are used in these conditions provide the interface to the trust service. A trust function will take a metric, a requester, a required score and possibly other arguments. It formulates a query to the trust service and compares the result to the required value and returns whether the required score is obtained. (Technically it does not return a Boolean but rather an EvaluationResult object for added generality.)

The main trust services used in this iteration are a reputation based (RTM) trust service, a credential based (CTM) service and a quality (in online compliance tests) based (OCTTM). The RTM service is described in detail in [D5.2]. The CTM service [D5.3] provides trust metric based on credential (chains). The POLIPO trust management system [TSZE09a, TSZE09b] forms the basis of this service. The service offers a SAML [SAML] compatible interface using SAML assertions to encode trust credentials. As SAML assertion requests do not cover all request formats supported by the trust management system, extensions to the request format are being considered. The credential cache implemented in an earlier iteration no longer needed as the new CTM service already stores non-local credentials.

The OCTTM service quality based trust by looking at the online compliance testing results [D10.3] obtained by a service.

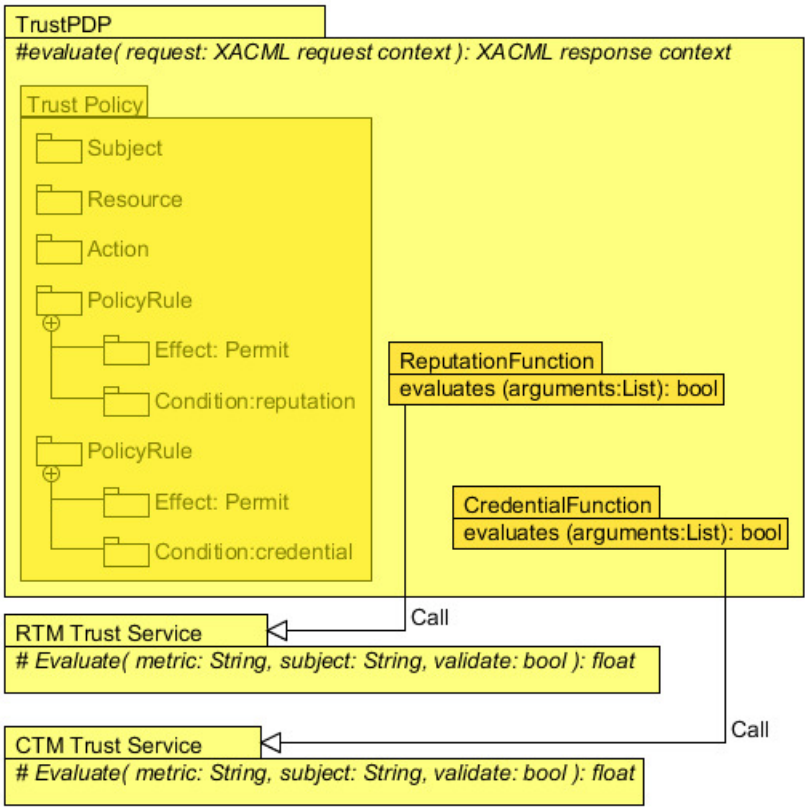


Figure 5.2 Trust PDP calling Trust Services

5.3 The Trust PDP

The trust PDP implementation extends Sun's (prototype) XACML v2.0 PDP implementation. By extending the `com.sun.xacml.PDP` object it provides a TrustPDP which is a PDP subtype which thus becomes useable in any context where a sun XACML PDP can be used. (Such as the PERMIS web service wrapper, see Section 3.2.4).

As already seen in examples in Section 3 and Section 4 and mentioned in Section 5.2 the overall trust policy is expressed using an XACML style policy set where conditions are used to express the trust requirements. E.g.

```
<Rule RuleId="id" Effect="Permit">
<Condition FunctionId=
"http://localhost:8080/trustpdp/names/function#reputation">
   <AttributeValue DataType=
   "http://www.w3.org/2001/XMLSchema#string">pagerank</AttributeValue>
```

describes that the reputation service needs to be called. The service call itself is provided by a `Function` object which exists for each supported trust service. `AttributeValue` elements are used to add the arguments that need to be passed to the Trust Service, such as the trust metric to use. (Note that the address is the one for

the function space of the Trust PDP not that of the reputation service; that is configured in the reputation function object itself.)

5.3.1 Functions and Trust Service calls

The `Function` object provides a wrapper around the trust service call. It implements the `com.sun.xacml.condition.Function` interface (typically be extending `eu.tas3.trustpdp.trustService.TrustMetric` or `eu.tas3.trustpdp.trustService.TrustFunction` which, in turn, extend `com.sun.xacml.condition.FunctionBase`) to enable trust metric evaluation to become part of the standard XACML policy evaluation framework.

As a trust service may offer different functions (e.g. for different types of metrics with different sets of arguments) the TrustService bundles the functions, implementing the `com.sun.xacml.condition.cluster.FunctionCluster` interface.

Below we describe the functions offered by the available trust service (RTM, CTM, OCTTM, GenericTM and SimTM). The functions typically come in two varieties; a trust functions returning a boolean and a trust metric returning a number representing a trust level. The name of the trust metric is obtained by adding the sub-namespace, which is equal to the name of the service in lowercase, to the name of the function. So for example rtm:avgFeedback is the metric corresponding with function avgFeedback. Note that, as described in Section 4.2.1, a trust metric has a fixed format for its arguments; the ratee as a string and an optional second string encoding all configuration information. The encoding of the configuration information into a string is given where relevant.

The currently supported functions are (omitting the name space prefix `http://localhost:8080/trustpdp/names/function#`);

`avgFeedback` – (provided by `eu.tas3.trustpdp.rtm.TrustService`). As an RTM trust function this takes two arguments; a string giving the entity that needs to be evaluated (ratee) and a number (double) that indicates the trust score required for this entity to be trusted. As an RTM trust metric is does not use any configuration information and returns the score of the ratee.

`reputation` – (provided by `eu.tas3.trustpdp.rtm.TrustService`). An RTM trust function for custom centrality measures which takes three arguments; a string given the centrality measure to use, a string giving the entity that needs to be evaluated and a number(double) that indicates the trust score required of this entity. The trust metric version of this accepts the centrality measure as its configuration information.

`credential` – (provided by `eu.tas3.trustpdp.ctm.TrustService`). A CTM trust function which takes three arguments describing the trust metric, i.e. the required credential (issuer (string or list), role (string), subject (string)) and a number with the required trust score i.e. 1 (optional, assumed to be 1 if missing. Recall that the CTM trust metric is binary, only returning 0 or 1). The trust metric version takes issuer:role as its configuration information.

`reputation-of-issuer` – (provided by `eu.tas3.trustpdp.rtm.TrustService`). A RTM trust function similar to `reputation` except that rather than check the reputation of the requester it find potential `issuers'

(e.g. of a credential) with a required reputation level. Typically used nested in another function (see e.g. 5.3.1.1). This function takes two arguments a string given the metric to use and a number (double) indicating the required score on this metric. (No trust metric version.)

`reputation-by-accredited-raters` – (provided by `eu.tas3.trustpdp.rtm.TrustService`). A RTM trust function which is similar to reputation except that it only takes into account the feedback from trusted `accredited' raters. Takes four arguments; a list of trusted raters (typically obtained from another trust function), the entity to be rated, the centrality measure to use and the required score on this metric. (A metric version of this can be obtained by using `rtm:reputation-among-trusted-raters`)

`rtm:reputation-among-trusted-raters`– (provided by `eu.tas3.trustpdp.rtm.TrustService`). An RTM trust metric which only considers feedback from trusted raters. Its configuration element metric.score specifies what score needs to be obtained on which metric to be considered trusted.

`credential-subject-list` – (provided by `eu.tas3.trustpdp.ctm.TrustService`). A CTM trust function similar to credential. It differs from this in the same way reputation-of-issuer differs from reputation; rather than checking if the requester has a credential this function finds all subjects that have a given credential. This function takes three arguments; the credential issuer and role and optionally the required trust score (i.e. 1).

`alwaystrue` – (provided by `eu.tas3.trustpdp.generictm.TrustService`). A testing trust function in which every entity is trusted. (Allows testing basic Trust PDP functioning without requiring access to any trust services.) Takes two optional arguments; a string describing the issuer and number (double) describing the required trust level. It always confirms that the subject has the required trust level. The closest to a trust metric version is generictm:fullyTrusted which returns 1 for all ratees.

`fullyDistrusted` – (provided by `eu.tas3.trustpdp.generictm.TrustService`). A testing trust function which always returns false. Takes two optional arguments; a string describing the issuer and number (double) describing the required trust level. The trust metric version always returns 0.

`random` – (provided by `eu.tas3.trustpdp.generictm.TrustService`). A testing trust function which returns a random trusted/untrusted decision for each call. Takes two optional arguments; a string describing the issuer and number (double) describing the required trust level. The trust metric version returns a random number in range [0,1].

`randomStatic` – (provided by `eu.tas3.trustpdp.generictm.TrustService`). Similar to random except that the ratee argument is required and all calls for the same ratee will give the same result.

`callBackRef` – (provided by `eu.tas3.trustpdp.generictm.TrustService`). A call back reference (explained in Section 5.3.2 below). This is a special function which takes as its argument a trust metric or function.

`AnyTested` – (provided `eu.tas3.trustpdp.octtm.TrustService`). A testing based trust function which trust an entity if any testing results have been reported for that

entity. This function enables trusting those with a testing policy in place (without looking at the actual results of the testing). It takes the ratee as its only argument. The trust metric version returns (the boolean) result as 0 or 1.

`TestCount` – (provided by `eu.tas3.trustpdp.octtm.TrustService`). A testing based trust function which looks at the percentage of successfully passed tests and compares this to a given threshold. (If not tests are found this is interpreted as failing to reach any threshold.) This function enables basic quality based trust. It takes the ratee and the required level as its arguments while the trust metric version returns the percentage of successful tests (with no tests interpreted as 0).

* Simulated metric – (provided by `eu.tas3.trustpdp.simtm.TrustService`). The simulation service is a testing tool which allows making available any trust metric. It allows setting a default value and chosen values for given ratees (see example in trustservices.xml, Section 3.2.1).

Trust Function / Trust Metric	**Arguments *=optional (Config string format)**	**Providing Service Object `eu.tas3.trustpdp.X.TrustService`**
avgFeedback	String ratee, Double score	rtm
reputation	String metric, String ratee, Double score	rtm
credential	String[] issuer, String role, String subject, Double score* (issuer:role)	ctm
reputation-by-accredited-raters/ rtm:reputation-among-trusted-raters	String[] raters, String ratee, String metric, Double score (metric:score)	rtm
alwaystrue/ fullyTrusted	String ratee*, Double score*	genericrtm
fullyDistrusted	String ratee*, Double score*	genericrtm
random	String ratee*, Double score*	genericrtm
randomStatic	String ratee, Double score*	genericrtm
callBackRef	Special (see Section 5.3.2)	genericrtm

AnyTested	String ratee	octtm
TestCount	String ratee, Double score	octtm
Nesting Functions (return type: String[])		
reputation-of-issuer	String metric, Double score	rtm
credential-subject-list	String issuer, String role, Double score*	polipotm

5.3.1.1 Nesting Functions

Several functions are specifically meant to be nested in others (e.g. reputation-of-issuer) or have others embedded (e.g. reputation-by-accredited-rater). However, others may also be nested. The nested functions need not be trust specific, any available function object can be used.

A nested function call is achieved by the Apply Function attribute, for example

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
and-only">
  <SubjectAttributeDesignator DataType=
  "http://www.w3.org/2001/XMLSchema#string" AttributeId=
  "urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
```

gets the identity of the subject (i.e. requester) as a string, which is useful for nearly all trust metrics.

The nesting of policies using specialized functions offers an efficient method of evaluating several classes of multilevel policies. However, it will always evaluate embedded metrics before calling the main function. This may not be the most efficient order of evaluation (or may not even be possible if the embedded metrics needs to be evaluated for specific ratees rather than looking for any trusted ratees). By using the call-back mechanisms, the evaluation of embedded metrics is postponed until it is actually needed. The following section describes how the call-back mechanism is supported in the trust PDP and what support is needed within the trust services and functions.

5.3.2 Call back Functions

The nested functions already offer the functionality of one trust metric depending on any other trust metric. By accepting a bag argument, the function can receive the set of entities which satisfy a minimum score on the other metric. Of course the nested metric must support being called in discovery mode [D5.1].

A disadvantage of this approach is that the nested metric must support discovery mode or the entities for which the embedded rating is needed must be known beforehand. Also, all nested metrics must be evaluated first. If we want to embed a metric which does not support discovery mode or running in discovery mode is too expensive from a performance point of view, e.g. because many different entities may be discovered but only the rating of a few is actually needed, then we need to use the call back mechanism. (Note that the call back mechanism causes its own overhead so one will need to consider whether it is really needed/beneficial.) In this case it would be better if the evaluation of the embedded metrics is postponed until it is known for which entities it is needed. This is what the call back function does; it specifies that the embedded metric should be registered for possible later evaluation rather than evaluated when encountered.

The call back functions is implemented throught a callback registry – this removes the need for sending embedded metrics to the trust services and returning them and has as a possible security benefit that the trust services can request call back evaluation for these metrics and not get access to arbitrary trust ratings, keeping the control over trust ranking release in the Trust PDP (though the trust service can request the rating of an arbitrary ratee on this metric).

The callback registry offers three main methods;
- `register`: takes an array of Evaluatable objects and returns a call-back-id.
- `evaluate`: which takes an set (implemented as an array or a bag) of ratees and a call-back-id and returns a set(array of bag) of ratee-score pairs (encoded as described above).
- `free`: takes a call-back-id.

The call back function will thus evaluate to a call-back-id, a reference to a metric and optionally a minimum level stored in the registry, instead of the value of the embedded metric. The main metric will have to determine when the call back is needed, and for which ratees the embedded metrics should be returned. It should then call the registry which will evaluate the reference for the given ratees and return the ratees which reach the minimum level, along with their scores. The answer is formatted as a bag of string of the format Ratee#Score. (As the score is a number the seperator # can be recognized as the last occurance so no additional restrictions is places on the format for Ratee.) This format is chosen to fit within the standard XACML types.

It is thus the responsibility of the main trust function to recognize the call-back, obtain the ratees which should be considered (likely be sending a special query to the trust service), passing these to the call back registry and with the result from the evaluation of the embedded metric compute the resulting score (likely by sending another query to the trust service). Another important responsibility of the trust service is to notify the register of any call back references which are no longer needed.

Function `reputation-among-trusted-raters` provides an example of how the call-back mechanisms can be used.

6 API and Library Information

6.1 Trust PDP Web Interface Wrapper

This wrapper uses the `XACMLAuthzRequest` WSDL operation definition of Kent's standalone authorization server [PERMIS], i.e. it accepts a XACML request context and responds with a XACML response context.

6.2 The Trust PDP

The Trust PDP interface provides a wrapper around the Trust PDP functionality.

```
/**
 * Get the PDP from the wrapper
 * @return The XACML PDP contained in this wrapper
 */
public com.sun.xacml.PDP getPDP();

/**
 * Get the name of this PDP
 * @return The string identifying this PDP.
 */
public String getPolicyIdentifier();

/**
 * Check whether this PDP is the default or not.
 */
public boolean isDefault();
```

The Trust PDP core is provided by the class eu.tas3.trustpdp.SimplePDP. It implements the interface above and also offers:

```
/**
 * Build a Trust PDP from a Trust PDP configuration element.
 * The configuration needs to give the policy configuration file and
 * the trust service configuration file. It may give a policy
 * identifier (otherwise the default `Trust PDP' is used).
 * @param e The element containing the configuration information
 *   for the Trust PDP. Must contain a PolicyConfigFile element and
 *   a TrustServiceConfigFile element. The (first) PolicyIdentifier
 *   element will be used if given. May contain a Ranking element;
 *   this consists of sub elements Service and Metric. Service is
 *   the class of the TrustService to use while ranking is the
 *   metric used to rank
 * @return The Trust PDP.
 * @throws Exception
 */
public static TrustPDPInterface buildPDP( org.w3c.dom.Element e ) throws
Exception;

/**
 * As buildPDP( Element e ) except that a (dynamic) trust Policy given
 * as a String may also be specified. This policy is added to the
```

```
 * policies described in the configuration element e.
 */
public static TrustPDPInterface buildPDP( org.w3c.dom.Element e, String
dynamicPolicy ) throws Exception;

/**
 * Main-line testing routine. This method lets you invoke the Trust PDP
 * directly from the command-line.
 *
 * @param args the input arguments to the class. They are either the
 *   flag "-config" followed by a configuration file and a request file,
 *   or a request file followed by one or more policy files (the default
 *   functions are used). In the case that the configuration file is
 *   used, the configuration file must provide a configuration element
 *  (see buildPDP).
 */
public static void main(String [] args) throws Exception;
```

6.3 The Service Interfaces

The CTM function cluster is implemented by the class
`eu.tas3.trustpdp.ctm.TrustService`

The RTM function cluster is implemented by the class
`eu.tas3.trustpdp.rtm.TrustService`

The OCTTM function cluster is implemented by the class
`eu.tas3.trustpdp.octtm.TrustService`

The basic function cluster is implemented by the class
`eu.tas3.trustpdp.generictm.TrustService`

The simulation function cluster is implemented by the class
`eu.tas3.trustpdp.simtm.TrustService`

No methods intended for external use exist in these objects.

6.4 The Trust Data Access Service Provider

The Trust Data Access Service Provider is obtained by providing a ranking service (see Section 6.2).

```
/**
 * Attempt to evaluate a trust request against the policies. If the
 * response is permit and a ranking service is set, and/or ranking
 * requests are specified in the request the rankings of
 * the requester are added to the result as additional sub-status
 * codes. (Each additional ranking will be a sub-status code of the
 * previous (sub)-status code to remain consistent with XACML; multiple
 * status codes are not allowed but arbitrarily deep nesting is.
 *
```

```
 * @param request the request to evaluate
 *
 * @return a response paired to the request
 */
public ResponseCtx evaluate(RequestCtx request);
```

7 License Information

The Software is provided under a BSD style license (license.txt):

"Copyright (c) 2009 Technische Universiteit Eindhoven.

All rights reserved.

This code uses Sun XACML code to which the license given in `license-sun.txt' applies.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the Technische Universiteit Eindhoven nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY Technische Universiteit Eindhoven "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Technische Universiteit Eindhoven BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

This code uses Sun XACML code to which the BSD style sun license applies (license-sun.txt):

"Copyright 2003-2004 Sun Microsystems, Inc. All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility."

8 Roadmap and Conclusions

The Trust tool set provides the functionality needed to support the use of trust requirements in the authorization and service discovery frameworks of the TAS³ architecture. It does this by providing a standard interface to the different trust services in the form of a Trust Policy Decision Point (Trust PDP). As such it is suitable to be used in the multi PDP setup, being called by a Master PDP, or as a standalone authorization decision point directly called by a Policy Enforcement Point (PEP).

This third iteration of the trust tool set offers a reference implementation which can be the starting point of building TAS³ trust network solutions. (Though it should be seen as a proof-of-concept, not a commercial level product.) It provides the following enhancement over previous iterations;

[Trust PDP] Support for updated trust services and a much more detailed template which makes supporting new trust metrics and services easier. Support for controlling order of evaluation within nested trust metrics using the callback mechanism.

[Trust Data Access Service Provider (TDASP)] Has inherited the improvements to the Trust PDP and adds a dynamic trust data querying mechanism. (Previously the type of trust data that would be available was decided statically.)

[Trust PDP webservice wrapper] Has been updated with new versions of the PERMIS PDP (component T3-PDP-P [PERMIS]) to maintain a consistent interface amongst the different PDPs in the project.

The trust PDP and TDASP offer the technical side of the trust management solution; they allow many factors of trust to be considered by supporting different trust services using different types and sources of trust information. To effectively deploy this trust management solution one needs to use appropriate trust metrics to describe the factors and offer the end user understandable choices on a suitable combination of factors. The trust of the end users, and thus the factors that should be considered and how these factors should be measured, depends on the application area. Thus, to improve effective use in the pilots and as a step towards further exploitation, we plan to look at end user trust perception in different application areas, including e-health and employability, and at methods to aid developers in creating suitable trust metrics. (On which we plan to report in [D9.2].)

9 References

[BEHH10] Böhm, K., Etalle, S., Hartog, J.I. den, Hütter, C., Trabelsi, S., Trivellato, D., Zannone, N. (2010). A flexible architecture for privacy-aware trust management. *Journal of Theoretical and Applied Electronic Commerce Research*, *5*(2), 77-96.

[D2.1] TAS³ Architecture, TAS³ deliverable.

[D5.1] Trust Management Architecture Design, TAS³ deliverable.

[D5.2] Behavioral Trust Management Engine, TAS³ deliverable.

[D7.1] Design of Identity management, Authentication and Authorization Infrastructure, TAS³ deliverable.

[D9.1] Pilots specification and use case scenarios, TAS³ deliverable.

[D9.2] Pilots Evaluation Report, TAS³ deliverable (2e iteration, to appear).

[D10.3] Intermediate OCT Report, TAS³ deliverable.

[PERMIS] PERMIS authorization system http://sec.cs.kent.ac.uk/permis/

[POLIPO] Polipo trust management system http://security1.win.tue.nl/~polipo

[SAML] Security Assertion Markup Language (SAML) 2.0 OASIS standard, http://saml.xml.org/saml-specifications.

[TSZE09a] Daniel Trivellato, Fred Spiessens, Nicola Zannone, and Sandro Etalle. Polipo: Policies & ontologies for interoperability, portability, and autonomy. IEEE International Workshop on Policies for Distributed Systems and Networks, 2009.

[TSZE09b] Daniel Trivellato, Fred Spiessens, Nicola Zannone, and Sandro Etalle. Reputation-based ontology alignment for autonomy and interoperability in distributed access control. In IEEE International Symposium on Secure Computing, Vancouver, CA, 2009.

[XACML] eXtensible Access Control Markup Language (XACML) OASIS standard, http://www.oasis-open.org/committees/xacml/

Amendment History

Ver	Date	Author	Description/Comments
0.1	2009-10-07		First draft
0.3	2009-10-08		WP internal version
0.5	2009-10-23		Version for internal review
1.0	2009-31-12		First iteration
1.1	2010-12-2		First complete draft 2nd iteration
1.3	2010-12-15		Version for internal review
2.0	2010-12-31		Second iteration
2.1	2011-5-25		First complete draft 3e iteration
2.2	2011-6-15		Third iteration draft for internal review
3.0	2011-6-27		Third iteration