SEVENTH FRAMEWORK PROGRAMME
Challenge 1
Information and Communication Technologies

Trusted Architecture for Securely Shared Services

Document Type:	Release Guide with Software Deliverable
Title:	**Back Office Services**
Work Package:	WP8
Deliverable Nr:	D8.2
Dissemination:	Final
Preparation Date:	December 23, 2010
Version:	2.0.1

The TAS³ Consortium

	Beneficiary Name	Country	Short	Role
1	KU Leuven	BE	KUL	Project Mgr
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	Coordinator
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	KIT	DE	KARL	Partner
20	Symlabs	PT	SYM	Partner

Contributors

	Name	Organisation
1	Marc Santos	University of Koblenz-Landau
2	Thomas Kirkham	University of Nottingham
3		
4		
5		

Contents

List of figures

1 Executive Summary

This document describes the "Back Office Services" used in TAS3. University of Nottingham and the University of Koblenz-Landau worked on the software bundle that provides a TAS3 adapted auditing system consisting of an Audit Bus and an Audit Service.

The Audit Service is a component developed by the University of Koblenz-Landau. It provides as back-end a database to store audit logs and as interface a web service that delivers and receives sets of audit entries. Access to the Audit Service is secured using TAS3 security components. Therefore Koblenz worked on the PEP component for the Audit Service that interacts with the Master PDP from Kent[1] and the zxid IDP [2].

The Audit Bus is a component developed by the University of Nottingham. Its main function is to relay messages regarding the trust fabric in the TAS3 network. For example the Audit Bus informs consumers of events related to service reputation, authorisation and authentication events specific to data objects in the TAS3 network. In this latest iteration the Audit Bus has moved from the use of Web Service SOAP interfaces and now is implemented usingAdvanced Message Queuing Protocol (AMQP) [3].

1.1 Reading Guide

There's one subsection for each software bundle/package that is described in this deliverable. The documentation of one package is independent from the other package. So the reader can understand the documentation of one package without having to read the other.

In the next chapter the reader finds details about the functionality of each of the software packages and their components.

Chapter 3 is about the installation and configuration of the described packages.

In chapter 4 the reader finds tutorials of how to use the software.

Chapter 5 gives some insights into the architecture of the components and chapter 6 is a short version of the API.

The last two chapters contain license information and a roadmap with functionalities that will be implemented in further versions of the described packages.

2 Introduction to the Software

2.1 Purpose

2.1.1 Audit Service

The Audit Service consists of 4 different components that handle the communication with the Audit Bus and offer Audit Service functions to call it from a client (SAWSService), provide a data structure for audit entries (Tas3LogRecordFormat), an abstraction layer on top of the database to store the audit trails (Tas3AuditTrail) and a graphical user interface to explore the audits (Tas3Notifications).

SAWSService and Tas3Notifications integrate TAS3 core components like the TAS3 Identity Provider and the Master PDP (see below).

TAS3 **Identity Provider** is the point where Users actually authenticate to the system. After authentication, the IdP issues a Single Sign-On (SSO) token so that the Front End Service can complete the login process. [2]

Master Policy Decision Point (Master PDP). The PEP calls Master PDP to obtain the **authorization** decision. All logic of the **authorization** decision is masked behind the Master PDP. Thus the exact implementation details of Policy Decision Point Stack are irrelevant for the PEPs [2].

2.1.1.1 SAWSService

SAWSService is a web service with the purpose of event auditing within the TAS3 infrastructure. Therefore, the service functionality can be invoked by any TAS3 component to save log entries containing information about events that occur whenever a document, web service or any other component is accessed. The type of messages can either be simple information, warnings or errors. Before sending an audit message to the SAWSService, the client has to authenticate by the TAS3 IdP and it has to be authorized by the TAS3 Master PDP.

2.1.1.2 Tas3LogRecordFormat

The log record format (Tas3LogRecordFormat[1]) used to save log records is derived from the OpenXDAS audit specification[2].

2.1.1.3 Tas3AuditTrail

In its current version, the audit trail data is persisted in a MySQL database (the Tas3AuditTrail provides the corresponding API to connect to the MySQL database).

---

[1] Tas3LogRecordFormat specification: http://tasssservices.uni-koblenz.de:8080/docs/tas3/SAWSService/download/Log%20Record%20Format.pdf

[2] OpenXDAS: http://openxdas.sourceforge.net/

2.1.1.4 Tas3Notifications

Tas3Notifications is a graphical user interface (GUI) that allows authenticated administrators and end users to browse through the audit trail and to discover events that have been logged within the Tas3 infrastructure. Its primary purpose is to help users get an overview about events related to objects (documents, data, etc.) that belong to that user, occurring within the TAS3 infrastructure.

2.1.2 Audit Bus

The purpose of the Audit Bus is to provide the TAS3 core infrastructure with a real time messaging component for critical cross architecture events. For example the change in a services trust ranking or an attempt by a service provider to access a specific data object.

The Audit Bus sends messages specific to critical events to parties that are both interested and authorised to receive them. The routing of the messages is done using a publish and subscribe architecture and is secured using certificate based encryption.

2.2 Scope

2.2.1 Audit Service

The scope of the Audit Service in TAS3 is to provide a TAS3 adapted and secured audit storage with a web service interface. In fact the TAS3 Audit Service is the only storage for audit data in the TAS3 demonstrations. Later on there will be no central Audit Service, but several providers for audit services [2]. So a TAS3 secured network can choose an audit provider and connect it to the chosen audit bus as a “Consumer” that receives audit messages and stores them.

2.2.2 Audit Bus

The scope of the Audit Bus is wide ranging and is the component that is used by most partners in TAS3. Communication with the bus occurs at key services in the trust network and particularly at the Policy Decision and Enforcement Point level. Clients that communicate with the Audit Bus are therefore implemented by service providers and the core infrastructure of the project. Other services that communicate with the Audit Bus include the Trust Management services provided by TUE and online compliance testing tool from CNR.

The Audit Bus links these communications to the user via the Dashboard and also the auditing service in TAS3. Therefore the structure of the Audit Bus has to handle a large amount of messages and process them with no failure. Thus for

the implementation of the Audit Bus reliable messaging technology has been chosen.

2.3 Functionality

2.3.1 Audit Service

2.3.1.1 SAWSService

The SAWSService web service is the core component encapsulating the functionality to save and retrieve log records after an authentication procedure. The required operations are exposed through a web service interface[3] as defined by the web services WSDL. Additionally, the SAWSService Application Programming Interface (API)[4] implements the required service stubs and provides a convenient way to access the SAWSService methods. The main features of the SAWSService are: Storing log records in the audit trail, retrieving the whole audit trail, searching for certain log entries. These features are implemented in the following web service methods:

`GetVersionResponse getVersion()`: Returns the version of SAWSService and log record format.

`ReadLogFileResponse readLogFile()`: Reads the whole log file and returns its content.

`SaveLogRecordResponse saveLogRecord(SaveLogRecordRequest logRecordRequest)`: Saves a log record containing the values specified by logRecordRequest. The class `SaveLogRecordRequest` is automatically generated from the web service's WSDL file (via WSDL2Java) and holds several attributes, one for each log record data field.

`SearchAuditTrailResponse searchAuditTrail(SearchAuditTrailRequest searchAuditTrailRequest)`: Searches in the audit trail and returns a list of log records that match the specified search parameters. The class `SearchAuditTrailRequest` is automatically generated from the web service's WSDL file (via WSDL2Java) and holds several attributes, one for each search parameter.

SAWSService relies on two other components, each providing an API that is used by the SAWSService: *Tas3LogRecordFormat* to handle log records, and *Tas3AuditTrail* to obtain the connection to the MySQL database.

2.3.1.1.1 SAWS-client

This component provides an API to access the SAWSService methods. It provides wrapper methods for the web service's method stubs that have been generated with the WSDL2Java[5] tool from the SAWSService WSDL document. Hence the

[3] See WSDL-Document: http://tasssservices.uni-koblenz.de:8080/axis2/services/SAWSService?wsdl

[4] See component SAWS-client introduced in the next section.

[5] Wsdl2java is a command line tool that is bundled with the Apache Axis2 distribution. See http://ws.apache.org/axis2/1_5/userguide-creatingclients.html for an introduction.

saws-client library provides a convenient way to use the SAWSService. It implements the following wrapper functions:

String `getVersion()`: Invokes the web service method `getVersion` and returns its response which is the SAWSService and log record format version.

String `parseAndSave(String logRecordString)`: Parses the given string representation of a log record and invokes the web service method `saveLogRecord` to save the record.

String[] `readLogFile()`: Invokes the web service method `readLogFile` which returns the content of the whole audit trail.

String `saveLogRecord(...<log record fields>...)`: Invokes the web service method `saveLogRecord` which saves the log record containing the given parameter values.

String[]: `searchAuditTrail(...<search params>...)`: Invokes the web service method `searchAuditTrail` and returns a string array containing the log records that match the specified search parameters.

Boolean **`authenticate`**`(String user, String role, String password)`: Calls the T3-IDP and T3-PDP to authenticate and authorize the originator of incoming or outgoing call.

2.3.1.2 Tas3LogRecordFormat

This encapsulates the aforementioned Tas3 log record format specification and provides an API to handle log records. The features of the Tas3LogRecordFormat API are:

- Generate new log records (as the Tas3 Log Record Format is a text-based format, i.e. data elements of type string separated by a '~', log records can either be generated by providing a full log record string, or by specifically providing the content of each data element).
- Check validity of existing log record strings.
- Access and edit data elements of existing log records.

As previously mentioned, the log record format used in the SAWSService is an adaptation of the OpenXDAS log record format specification. Hence, the Tas3Log-

RecordFormat syntax is similar to the OpenXDAS specification. The semantic of data elements is defined in the Tas3LogRecordFormat specification[6].

The Tas3LogRecordFormat API provides a set of java classes that represent the data elements of a specific log record and provide the functions for the aforementioned features. Basically, a specific log entry is a string containing string values for each log record field separated by '~'. The following lines show a template of a SAWSService log record (the elements HDR, ORG, INT, TGT, SRC, EVT, END are fixed elements dividing each log record into six blocks of data):

```
HDR~
hdr_format_version~
hdr_date_time~
hdr_time_uncertainty_interval~
hdr_time_uncertainty_indicator~
hdr_time_source~
hdr_time_zone~
hdr_event_number~
hdr_event_outcome~
ORG~
org_location_name~
org_location_address~
org_service_type~
org_auth_authority~
org_principal_name~
org_principal_id~
INT~
int_auth_authority~
int_domain_specific_name~
int_domain_specific_id~
TGT~
tgt_location_name~
tgt_location_address~
tgt_service_type~
tgt_auth_authority~
tgt_principal_name~
tgt_principal_id~
SRC~
src_pointer_to_source_domain~
EVT~
evt_event_specific_information~
END
```

The first 6 header fields are automatically set whenever a new log record is created (the fields `hdr_time_uncertainty_interval`, `hdr_time_uncertainty_indi-cator` and `hdr_time_source` are left empty due to minor relevance). All remaining fields are optional and have to be set by the user. The Tas3LogRecordFormat specification[7] provides more information about the data elements, mappings of event numbers (data element `hdr_event_number`) and event outcome codes (data element `hdr_event_outcome`) to human readable strings as well as templates for valid log record strings.

[6] http://tasssservices.uni-koblenz.de:8080/docs/tas3/SAWSService/download/Log%20Record%20Format.pdf

[7] http://tasssservices.uni-koblenz.de:8080/docs/tas3/SAWSService/download/Log%20Record%20Format.pdf

2.3.1.3 Tas3AuditTrail

This encapsulates the functionality to obtain the database connection and to persist and retrieve log record data from the database. This component provides the interface between applications and the data store. In the current release, log records are stored in a MySQL database. Storing log records via the component SAWS (developed by project partner University of KENT) is basically covered by the Tas3AuditTrail API, but is unused due to performance pitfalls of SAWS and current ongoing development by KENT. KENT is working on those performance issues so that the Tas3AuditTrail will be replaced using their service.

This component uses Hibernate[8] for the Object/Relational Mapping (ORM) between the data elements of the Tas3LogRecordFormat to the data fields of the corresponding MySQL database table. In order to save or retrieve log records, the Tas3AuditTrail API provides suitable functions. Detailed information about classes and functions can be found in the Tas3AuditTrail javadoc.[9]

2.3.1.4 Tas3Notifications

Tas3Notifications is a web application that allows users and administrators after an authentication procedure to show audit trails and search for specific events that have been logged. This application only reads the audit trail by using the SAWSService methods to retrieve the full audit trail or to search for specific log entries. Tas3LogRecordFormat is furthermore used to handle the log records retrieved from the audit trail data store.

Implementation platform:

- Enterprise Java Beans 3: Business logic.
- JBoss Seam 2.1: Framework for communication between clients and business logic.
- JBoss Richfaces 3.3: Java Server Faces based framework for the development of rich user interfaces.
- JBoss Application Server 4.2.3 / 5.0.1: Application Server.
- TAS3 Master PDP - T3-PDP-M
- TAS3 IdP - T3-IDP

[8] http://www.hibernate.org/

[9] http://tassservices.uni-koblenz.de:8080/docs/tas3/SAWSService/javadoc-tas3audittrail/index.html

2.3.2 Audit Bus

2.3.2.1 Overview
When a user joins a TAS3 trust network they are presented with a Dashboard to provide them with key events about the use of their data in TAS3 enabled applications. The Dashboard functions by receiving event updates specific to the user via the Audit Bus.

Although the user does not see the Audit Bus it carries all the logging data that relates to the use of the users data on the system. The user retrieves this data via either the Dashboard or interfaces to the TAS3 logging services both of which are fed by the Audit Bus.

In the latest AMQP implementation the Audit Bus consists of a main message Broker. This Broker provides the key functions of the Audit Bus.

2.3.2.2 Message Subscription
The messages on the Audit Bus are sent down specific channels in order to separate them in terms of relevance. Services either publish or subscribe to messages from the Bus on these channels. The list of channels can be seen in Appendix 1.

When subscription to a channel is requested the certificate used to sign the request is checked by the Audit Bus broker this allowing only trusted parties interaction with the bus. The messages sent from the broker are also SSL encrypted.

Once subscribed the Audit Bus client can receive all updates from the channel that they have subscribed to. The subscribers to the Audit Bus agree to strong rules regarding the information they receive from the Audit Bus and only strip away the data that is relevant to them.

2.3.2.3 Message Publication
In the same way as subscriptions to the Audit Bus are doing using SSL based authentication, so is the publication of messages to the Audit Bus.

In terms of personal data the events are gathered from the policy enforcement and decision points in the trust network. These points invoke the Audit Bus directly as shown in Figure 1.

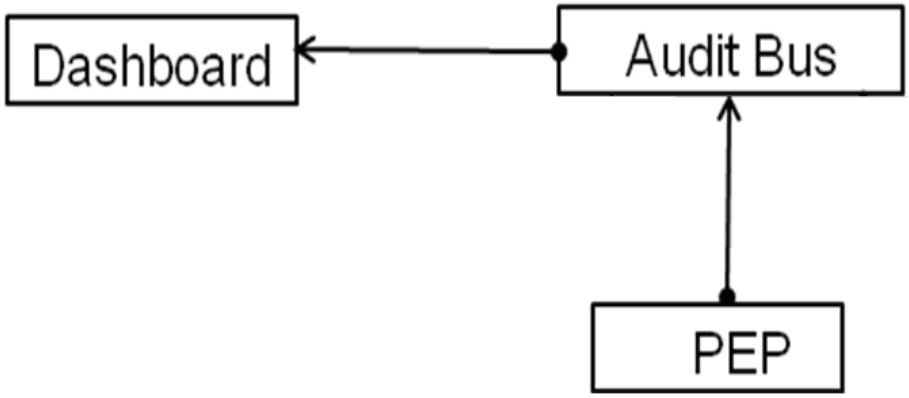

Figure 1: Message Publication.

The figure above illustrates the Audit Bus links the Dashboard to the policy enforcement infrastructure. Typical messages in this case are usually related to service provider access to TAS3 secured data objects. Here the result of the request is passed by the PEP to the data owner (usually the user) to the Dashboard.

2.4 Available Releases and Components

The TAS³ software pool contains the following releases:

2.4.1 Audit Service

2.4.1.1 SAWSService

Component	Version	Comments
T3-LOG-WRAP-SAWS	0.9	TAS3 Audit Service. Remark: This package also contains the components "Tas3LogRecordFormat" and "Tas3AuditTrail"

2.4.1.2 SAWS-client

Component	Version	Comments
T3-LOG-GUI	0.7	A desktop application to visualize audit data.

2.4.1.3 Tas3LogRecordFormat

This component is included in the "SAWSService" component.

2.4.1.4 Tas3AuditTrail

This component is included in the "SAWSService" component.

2.4.1.5 Tas3Notifications

Component	Version	Comments
T3-DASH	0.4	A web based interface for end users to overview audit data.

2.4.2 Audit Bus

There are two releases of the Audit Bus. The release that was used in the last review was implemented using Apache MUSE. This version is SOAP based and is a Java implementation of the Web Services Resource Framework. The MUSE implementation consisted of a main Audit Bus web service hosted using Apache Tomcat. The service provides publish and subscribe mechanisms for notifications

from the Audit Bus. However the MUSE Audit Bus is no longer used for Audit events and will be used to carry management events in the project.

The current 0.4 T3-BUS-AUD component release is implemented using Apache QPID. This is an Open Source implementation of the Advanced Messaging Queuing Protocol. The Audit Bus key component is a AMQP broker that has been implemented using C++. The Audit Bus supports AMQP protocol versions 0-10 and can be communicated to using clients written in Java, Python, Ruby and .Net.

3 Installation Guidelines

3.1 Hardware and Software Prerequisites

3.1.1 Audit Service

In order to deploy and run the software components, your system needs to meet the following requirements:

- Java Runtime Environment (JRE) and Java Development Kit (JDK), minimum Version 6.
- Apache Tomcat web server, minimum Version 6.0.14.
- Axis2 running inside the Apache Tomcat servlet container.
- MySQL database server, minimum Version 5.0
- JBoss Application Server, minimum Version 4.2

TAS3 related prerequisites:

- TAS3 Master PDP - T3-PDP-M
- TAS3 IdP - T3-IDP

3.1.2 Audit Bus

Apache QPID can run on both Windows and Linux. The software is available in precompiled formats or as source. In order to build QPID you will need Apache Ant in order to run the build scripts.

3.2 Installation and Configuration Instructions

3.2.1 Audit Service

The software components Tas3LogRecordFormat, Tas3AuditTrail, BusConsumerClient and SAWS-client are Java libraries providing APIs to access their functionality. They can be used by copying them into lib-directory of the Java application which will access the functionality.

The following section focuses on the most important information required to install and deploy the software applications and set up the MySQL database.

All components can be found in the component pool in the TAS3 portal https://portal.tas3.eu/pool/ or in our Subversion repository under https://svn.uni-koblenz.de/mpinto/tas3 (user = tasmember – password = svnTas3!)

3.2.1.1 SAWSService

Copy SAWSService.aar into the folder AXIS2_ROOT/WEB-INF/services and start the Apache Tomcat web server. The SAWSService web service should now be running.

3.2.1.2 MySQL database set up

The MySQL database set-up has to conform to the Tas3AuditTrail hibernate configuration. Both the corresponding database and user have to exist:

Database: tas3audittrail
User: tas3audittrail (username), s3cure (password)

Hibernate automatically creates the database table from the MySQL entity defined in de.uniko.iwm.tas3.audittrail.persistence.mysql.MySQLRecord.java once Tas3AuditTrail is compiled for the first time.

3.2.1.3 SAWS

See http://sec.cs.kent.ac.uk/permis/downloads/Level1/SAWS.shtml for installation details of SAWS.

3.2.1.4 Tas3Notifications

To deploy the Tas3Notifications web application, copy Tas3Notifications.ear to the deploy directory of the JBoss application server.

3.2.2 Audit Bus

The specific broker used in the project can be downloaded from the component pool in the TAS3 portal https://portal.tas3.eu/pool/ . The client files can also be downloaded from here.

Once downloaded and extracted the bus Java client setup can be build by running the command ant in the AuditBusQPID root directory of the download. This should build the broker and the test applications.

The broker is uses the C++ version of Apache QPID and is built by using the command `./configure` and `make all` in the root directory. More information on building the QPID broker in C++ can be found on the Apache QPID website here http://qpid.apache.org/getting_started.html**.** In order to configure SSL encryption you need to create and reference a SSL key store.

3.3 Running the Tests

3.3.1 Audit Service

The best way to test the Audit Service is by using the SAWS Service Client API (see section 4.1.1.2). This client API provides the main functions to send and to receive messages to and from the audit service. Prerequisite is the correct

installation of the audit service and a known endpoint URL to call the service. Moreover the client has to be authenticated to the TAS3 IdP.

The following code describes a simple JAVA based console program that instantiates a SAWSClient and looks in the audit storage for audits that happened during a given period.

```
// import the Tas3LogRecordFormat package
Import de.uniko.iwm.tas3.logrecordformat.*;

// create a new client
SAWSClient sawsClient = new SAWSClient(EPURL);

String endDate = LogRecord.formatDate(2009,10,16,23,00,00);
String[] searchResults = sawsClient.searchAuditTrail(
      "and","","","",endDate,"","","","","","","","","","","","
      ","","","","","","","","");
```

In this example the function formatDate is used to format the date time string. The return value of this function call is "2009-10-16 23:00:00". (see section 4.1.1.2. for more details)

The following code fragment creates an audit object, serializes it and sends it to the audit service.

```
String response = sawsClient.saveLogRecord("hdr_event_number",
    "hdr_event_outcome",
    "org_location_name",
    "org_location_address",
    "org_service_type",
    "org_auth_authority",
    "org_principal_name",
    "org_principal_id",
    "int_auth_authority",
    "int_domain_specific_name",
    "int_domain_specific_id",
    "tgt_location_name",
    "tgt_location_address",
    "tgt_service_type",
    "tgt_auth_authority",
    "tgt_principal_name",
    "tgt_principal_id",
    "src_pointer_to_source_domain",
    "evt_event_specific_information");
```

As test result each of the called methods send back an answer: “0” is sent back in the case of a successful call and a successful execution of the procedure. “1” would indicate a failure.

More detailed information about integration tests can be found in Deliverable D12.2 [5]. The Audit Service is part of almost every test case described in this "Trust & Application Integration Report".

3.3.2 Audit Bus

In order to start the tests the broker must be started. This can be done by running the following script located in the Audit Bus code from the portal.

```
/qpidd --load-module .libs/ssl.so --ssl-cert-password-file passwd --ssl-cert-db /CA_db --ssl-cert-name MyRootCA
```

The script above starts the broker with SSL enbled but you can also invoke the broker on an unsecured port if it is not blocked. Normally the secured port is 5671 and the unsecured port is 5672.

```
[root@kirkhab1 src]# ./qpidd --load-module .libs/ssl.so --ssl-cert-password-file
 passwd --ssl-cert-db /CA_db --ssl-cert-name MyRootCA
2010-11-04 12:40:17 notice SASL disabled: No Authentication Performed
2010-11-04 12:40:17 notice Listening on TCP port 5672
2010-11-04 12:40:17 notice Listening for SSL connections on TCP port 5671
2010-11-04 12:40:17 notice Broker running
```

The test clients are Publisher and Subscriber clients. These are implemented in Java and can be run using a script `runTests.sh`. The example clients can be downloaded from here https://portal.tas3.eu/pool/ and should be extracted onto the server.

The tests script is located in the directory
`/AuditBusQPID/TAS3/2011/src/main/java`

In order to run the test script the QPID_HOME variable needs to be set. This should reference the root directory of a QPID Java installation that can be downloaded from the Apache QPID web site.

Once the broker is started the clients can be run. The subscriber should be started first from the java dir of the examples using

`./runTests.sh` org.apache.qpid.tas3.auditbus.Subscriber

The client establishes a connection with the broker and listens for notifications for a specific topic. The topic can be changed by modifying the source code related to the Subscriber.

```
[root@kirkhab1 java]# ./runTests.sh org.apache.qpid.TAS3.AuditBus.Subscriber
Using QPID_HOME: /usr/share/AuditBusQpid/AuditBusQPID/
Using QPID_SAMPLE: /usr/share/AuditBusQpid/AuditBusQPID/TAS3/2011/src/main/java
/main/java
Received msg:msg:1
```

The publisher client is started by running the same script specifying that the Publisher should run. The publisher sends out a message on a topic and the subscriber should receive it. This is done by running the command

`./runTests.sh` org.apache.qpid.tas3.auditbus.Publisher

The publisher communicates with the broker specifying a topic on what it wants to post on. The broker can validate the publisher is the correct entity by checking the certificate and a access control list is also applied to the requested post on the topic.

See the images below.

```
[root@kirkhab1 java]# ./runTests.sh org.apache.qpid.TAS3.AuditBus.Publisher
Using QPID_HOME: /usr/share/AuditBusQpid/AuditBusQPID/
Using QPID_SAMPLE: /usr/share/AuditBusQpid/AuditBusQPID/TAS3/2011/src/main/java
Sent:1
Done.
```

For further information on the activity of the broker it can be started in logging mode. This is done by specifying –t after the broker start-up string.

4 How to Use the Software

4.1 Tutorial

4.1.1 Audit Service

4.1.1.1 Invoking the SAWSService methods

The service stubs to invoke the SAWSService methods can be automatically generated from the service's web service description file (SAWSService.wsdl) by using the WSDL2Java tool. The important classes required to invoke the web service methods are:

- SAWSServiceSAWSServiceHttpSoap11EndpointStub - Stub class for remote method invocation.
- SaveLogRecordRequest - Used for defining the log record values.
- SaveLogRecordResponse - Response sent back to the client after saving a log record.
- ReadLogFileResponse - Response sent back to the client after reading an audit trail.
- SearchAuditTrailRequest - Used for defining the search parameters.
- SearchAuditTrailResponse - Response after searching containing search results.

The following listing shows the important prerequisites required to use the service stubs:

```
// import the package to which WSDL2Java has generated the
service stubs and classes
import de.uniko.iwm.tas3.saws.service.*;
...
// define the service endpoint url
private static final String EPURL =
"http://tasssservices.uni-
koblenz.de:8080/axis2/services/SAWSService";
...
// create a new service stub
SAWSServiceSAWSServiceHttpSoap11EndpointStub serviceStub =
    new SAWSServiceSAWSServiceHttpSoap11EndpointStub(EPURL);
```

The object serviceStub provides the web service methods. The following listing shows how to invoke a web service method and how to get the response by using the corresponding getter method.

```
// call getVersion method and get response
String version = serviceStub.getVersion().get_return();
```

Invoking the other web service methods works similarly. The methods saveLogRecord and searchAuditTrail additionally require the parameters SaveLogRecordRequest and SeachAuditTrailRequest, each providing getter and setter methods for their attributes.

```
SearchAuditTrailRequest searchAuditTrailRequest =
      new SearchAuditTrailRequest();

searchAuditTrailRequest.setSearch_hdr_date_time_end("2009-10-
16 23:00:00");

// define other search parameters here by using the setter
// methods of searchAuditTrailRequest

// invoke web service method method
SearchAuditTrailResponse searchAuditTrailResponse =
    serviceStub.searchAuditTrail(searchAuditTrailRequest);

// get response's lines
String[] searchResult = searchAuditTrailResponse.get_return();
```

This example searches the audit trail for log records saved before Friday, October 16th 11 PM. You can specify other search parameters for a more detailed search. Note: It's important to define a datetime value in the format used in this example, otherwise the service will throw a ParseException while trying to parse the datetime string value. The class LogRecord of the Tas3LogRecordFormat (see Javadoc) provides the static function formatDate which should be used to format datetime values.

4.1.1.2 Using the SAWS Service Client API

A more convenient way to access the SAWSService methods is provided by the SAWSService client API. Remote method invocation is simplified through wrapper functions which manage the correct instantiation of required stubs and classes. The equivalent for the search example above is illustrated in the following listing.

```
// import the Tas3LogRecordFormat package
Import de.uniko.iwm.tas3.logrecordformat.*;

// create a new client
SAWSClient sawsClient = new SAWSClient(EPURL);

String endDate = LogRecord.formatDate(2009,10,16,23,00,00);
String[] searchResults = sawsClient.searchAuditTrail(
```

```
"and","","","",endDate,"","","","","","","","","","","","
","","","","","","","","";
```

In this example the function formatDate is used to format the date time string. The return value of this function call is "2009-10-16 23:00:00". Defining empty string values for the other parameters has the effect that corresponding log record values will be ignored during the search process.

Saving a log record works similar. The function saveLogRecord handles the remote method invocation to save a log record specified by its parameters.

```
String response = sawsClient.saveLogRecord("hdr_event_number",
    "hdr_event_outcome",
    "org_location_name",
    "org_location_address",
    "org_service_type",
    "org_auth_authority",
    "org_principal_name",
    "org_principal_id",
    "int_auth_authority",
    "int_domain_specific_name",
    "int_domain_specific_id",
    "tgt_location_name",
    "tgt_location_address",
    "tgt_service_type",
    "tgt_auth_authority",
    "tgt_principal_name",
    "tgt_principal_id",
    "src_pointer_to_source_domain",
    "evt_event_specific_information");
```

This function call constructs a string using the SAWSService log record format and calls the SAWSService method to save the log record. The response sent back to the client indicates success ("0") or failure ("1").

Each remote procedure call has to be verified. Therefore the T3-PDP and T3-IDP have to be called. All messages are secured using the TAS3 ZXID-AXIS filter.

4.1.1.3 Tas3LogRecordFormat

Tas3LogRecordFormat provides functionality to create and process log record entries.

In order to create a new log record, a new LogRecord object has to be created. To provide the log record values already on construction time, the different log record values can be provided as parameters of the contructor:

```
LogRecord record = new LogRecord(
    hdr_event_number, hdr_event_outcome,
    ...
    evt_event_specific_information);
```

Alternatively, the constructor can be invoked without parameters leading to a new empty LogRecord object. This object can be filled with an existing log record by invoking the objects parse method

```
record.parse(logRecordString);
```

with logRecordString being a string complying to the Tas3 Log Record Format specification. This method can be used for instance after retrieving a log record string from the MySQL database via Tas3AuditTrail.

The object's `toString()` method can be used to print out the log record as a string.

4.1.1.4 Tas3AuditTrail

Tas3AuditTrail provides functionality to retrieve the Tas3 audit trail, to persist log records as well as to search for specific log entries.

First, an object of Tas3AuditTrail has to be created specifying the desired persistence mode, i.e. where log records shall be stored (MySQL or SAWS):

```
AuditTrailClient auditTrailClient =
        new AuditTrailClient(PersistenceMode.MYSQL);
```

The whole audit trail can be retrieved by invoking

```
auditTrailClient.getAuditTrail();
```

This returns an array of String, each array element representing one log record in the audit trail (such a log record string can be processed by using the Tas3LogRecordFormat functionality as described in the previous section).

In order to save a given log record, the object's saveAuditTrail method has to be invoked

```
auditTrailClient.saveAuditTrailRecord(record);
```

with `record` being a string complying to the Tas3 Log Record Format.
To search for specific log records, e.g. log records stored between two given dates, the search method has to be invoked. The parameters thereby define the search parameters against which log records in the audit trail will be matched. Positive matches will be returned in an array of log record strings.

```
String[] searchResults = searchAuditTrail(...);
```

4.1.1.5 Tas3Notifications

Tas3notifications is a graphical user interface that is part of the TAS3 Dashboard. The following paragraph shows some main screens of this user interface and describes the functionality.

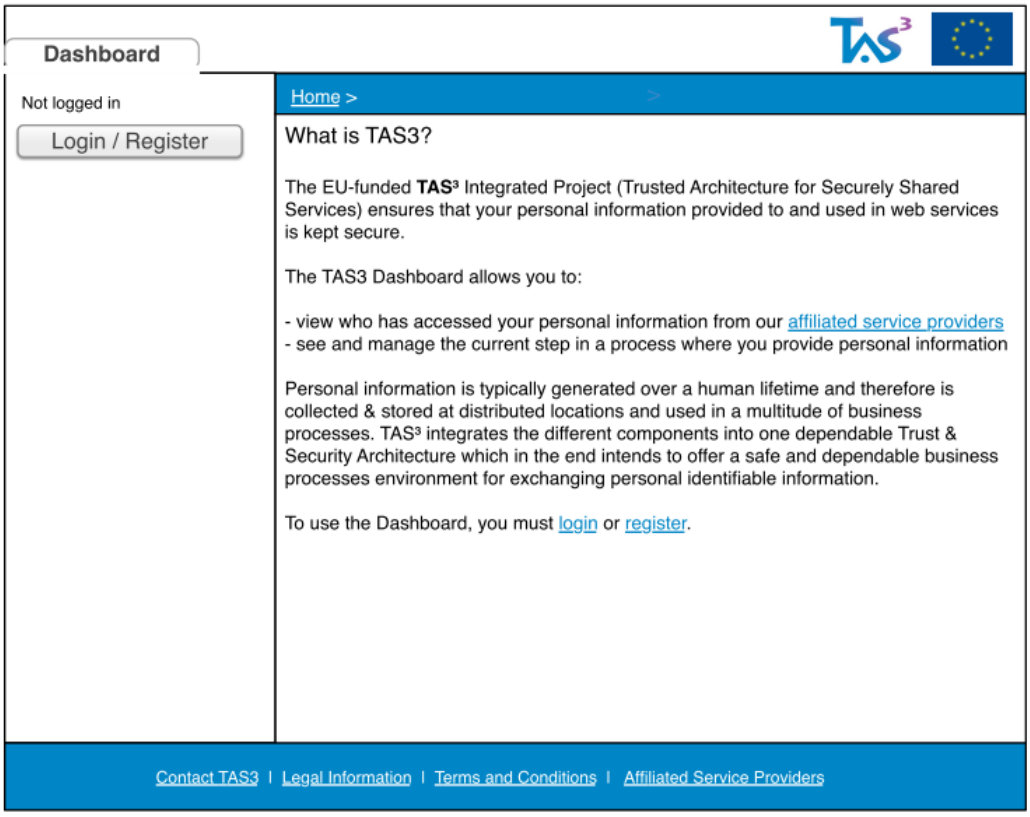

Figure 2: Start page of the TAS3 Dashboard

In the above figure the reader can see the start page of the TAS3 Dashboard. In this case the user is not logged in, so the some general info is shown in the main content area. Moreover some utility navigation is shown in the footer menu.

Clicking on LOGIN/REGISTER loads Remote Login screen in the main content. So the zxid IdP authentication page is called.

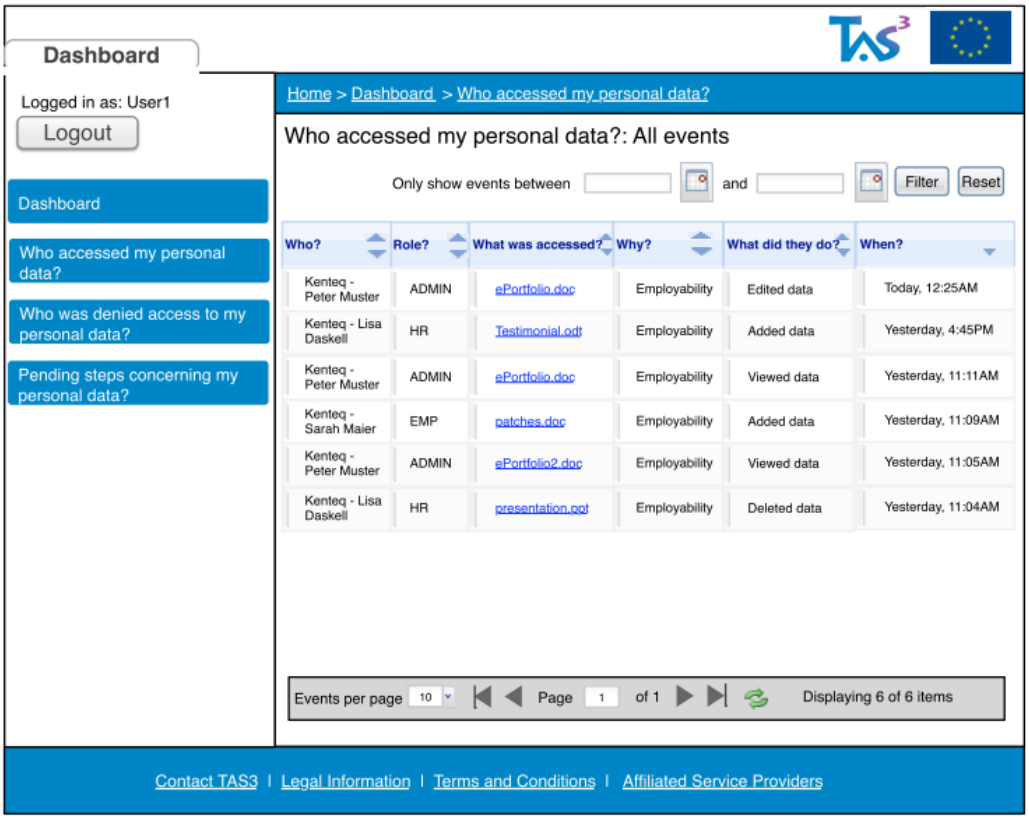

Figure 3: Audit Viewer showing results of the query: Who accessed my personal data?

- Default view is to show all successful events in descending date order; only show data access by others (not including end user). User can filter events to be shown between two dates; heading is updated to show the displayed date range (e.g. events between <date1> and <date2>). (see above figure)
- Reset returns user to view all events.
- Show data access events in sortable columns, default sort order is descending date from most recent events.
- User can scroll through events, using controls. Grey out unavailable options. Number of events shown per page can be set by user (option to show 5, 10, 25, 50 and 100).
- Right-hand container is scrollable independently of left-hand menu container.
- Click on resource goes to Data Detail screen.

Deliverable D8.3.[4] provides more information about the usage of graphical user interfaces in work package 8.

4.1.2 Audit Bus

In order to set the broker up and do basic tests the steps in section 2 can be followed. However in the project the interaction with the broker will be on different channels and will use SSL encryption, the following guide will be of assistance.

Transport Level Security

Transport level security is achieved by the use of SSL certificates. A SSL certificate can be generated using generic tutorials on the web, a qpid specific one can be found here http://rajith.2rlabs.com/2010/03/01/apache-qpid-securing-connections-with-ssl/

Simple SSL

In the testing scenario the broker is started to support SSL encryption. In this string a local certificate database is needed for use by the broker. In the startup string this is specified by the `–ssl-cert-db` option, the `–ssl-cert-name` option specifies the name of the certificate. If the database is password protected the –`ssl-cert-password-file` option should specify the location of a file that will contain the password. On the client side the public key of the certificate used by the broker needs to be known.

Client Authenticated SSL

In order to verify the client the clients public key needs to be shared with the broker. The broker is supported to use client authentication by supplying the following option in the command line `--ssl-require-client-authentication.` When this option is supplied the client will be asked for the public key during the TLS data exchanges.

Access Control

Access Control lists can also be specified when starting the broker and add an extra layer of protection. Here the ACL checks the

Modifying Topics

Topics are the names of event channels supplied by the bus. A list of topics used in TAS3 can be seen in Appendix1. In order to modify the topics that a client uses a small modification is needed to the client code. This can be done by editing this file:
`AuditBusQPID/TAS3/2011/src/main/java/org/apache/qpid/TAS3/AuditBus/ConnectionSetup.java`

Inside the file the topic is specified in the line of code below:

```
final static String TOPIC_NAME = "SRC";
```

Once the file is edited it can be recompiled by using `the ant` command in the directory: `AuditBusQPID/TAS3/2011/src/main/java`

5 Architecture

5.1 Audit Service

The Audit Service belongs to the TAS3 User Tools and is related to the "Dashboard", which is an user front-end where the user can overview all her data, data related transactions and other users who tried to access the user's data (section 4.1.1.5. in this document and Deliverable 8.3. shortly describe the dashboard [4]).

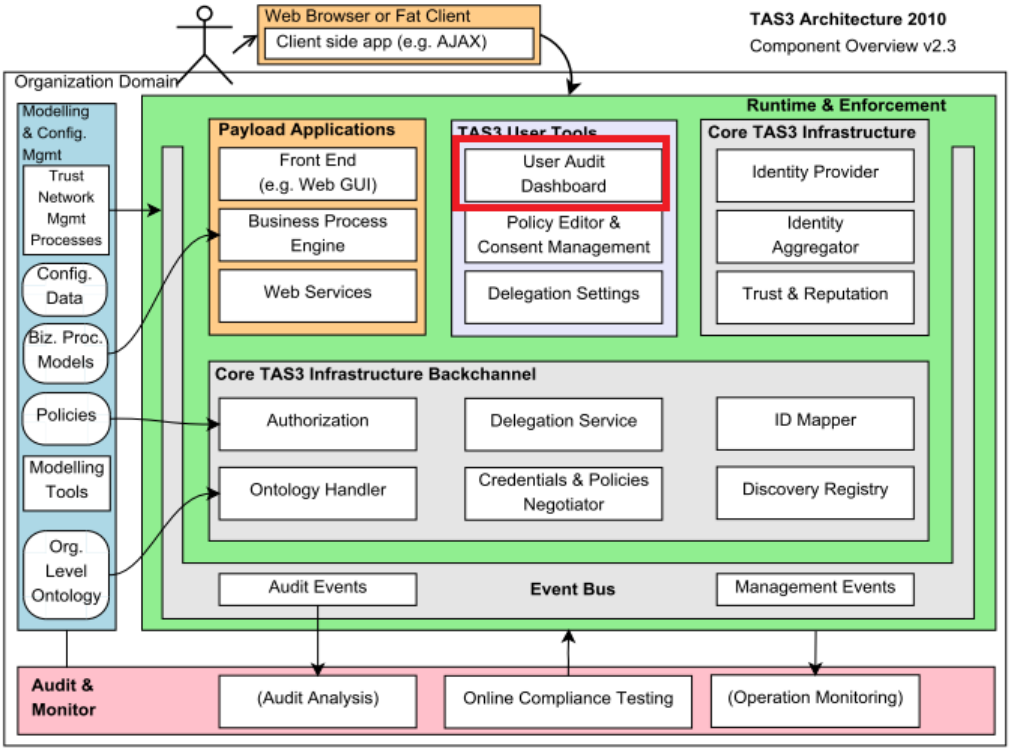

Figure 4: Position of the Audit Service in the TAS3 Architecture

When the Dashboard functions as a front-end for the audit service, the Audit Bus is one of the non end user clients that interacts with the Audit Service to send audit messages to it. It is possible to send directly messages to the Audit Service to store them, but in the demonstrators /pilots the Audit Bus is the main client that calls the Audit Service.

5.2 Audit Bus

Wider TAS3 Architecture

The Audit Bus is a infrastructure specific service within TAS3. It is provided to ensure that the TAS3 trust network operates in a secure and reliable way by the provision of real-time messaging. As discussed the Audit Bus carries event

information regarding access requests to data objects and other user / service provider behaviour on the network.

The Audit Bus therefore receives messages from many components in TAS3 and accepts messages in the AMQP format. The structure of the messages is further determined by the type of message and its original source. The Audit Bus does not read or handle the message content in anyway and only forwards events based on their AMQP channel.

Security is provided in the Audit Bus using the SSL and Client Authentication at transport level. Each invocation is also checked to make sure it corresponds to a valid client. At application level the services that receive messages from the Audit Bus have to authenticate with the TAS3 network and therefore have the right privileges to receive the Audit Bus messages.

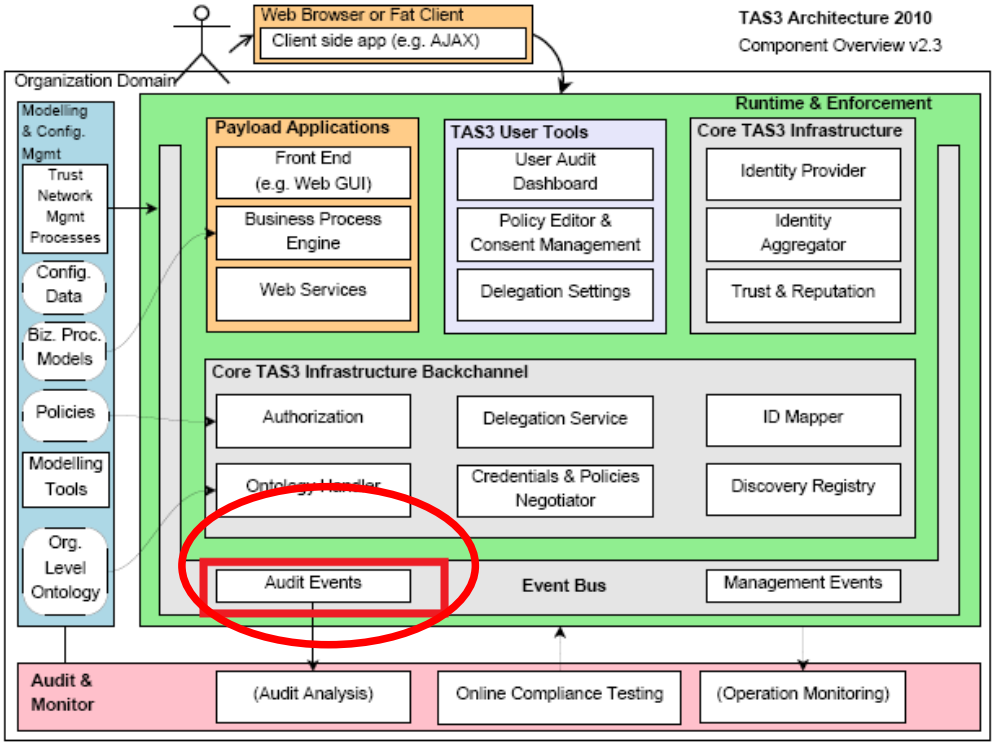

Figure 5: Position of Audit Bus in TAS3 Architecture.

The Audit Bus operates using a publish and subscribe mechanism of eventing.

6 API and Library Information

6.1 Audit Service

The following content is an extract of the Audit Service API. As mentioned before, this AXIS web service has a TAS3-AXIS filter enabled. This filter secures all outgoing and incoming messages using zxid. So all remote procedure call you see below need an authentication and authorization step before they can be executed.

parseAndSave

public java.lang.String **parseAndSave**(java.lang.String logRecordString)

Parses the given string representation of a log record and invokes the web service method saveLogRecord to save the record.

Parameters:
logRecordString - String representation of a log record. Must conform to the specified log record format.

Returns:
0 if saving the log record succeeded, otherwise 1.

saveLogRecord

public java.lang.String **saveLogRecord**(java.lang.String hdr_event_number,
java.lang.String hdr_event_outcome,
java.lang.String org_location_name,
java.lang.String org_location_address,
java.lang.String org_service_type,
java.lang.String org_auth_authority,
java.lang.String org_principal_name,
java.lang.String org_principal_id,
java.lang.String int_auth_authority,
java.lang.String int_domain_specific_name,
java.lang.String int_domain_specific_id,
java.lang.String tgt_location_name,
java.lang.String tgt_location_address,
java.lang.String tgt_service_type,
java.lang.String tgt_auth_authority,
java.lang.String tgt_principal_name,
java.lang.String tgt_principal_id,

java.lang.String src_pointer_to_source_domain,

java.lang.String evt_event_specific_information)

Invokes the web service method saveLogRecord which saves the log record containing the given parameter values.

Returns:
0 if saving the log record succeeded, otherwise 1.

readLogFile

public java.lang.String[] **readLogFile**()

Invokes the web service method readLogFile which returns the content of the whole audit trail.

Returns:
A string array containing the log records of the audit trail.

setEpUrl

public void **setEpUrl**(java.lang.String epUrl)

Sets the endpoint Url of this SAWS Client to the specified value.

Parameters:
epUrl - - the endpoint url which is the url pointing to the location where the SAWSService is deployed.

getEpUrl

public java.lang.String **getEpUrl**()

Returns the endpoint Url of this SAWS Client.

Returns:
The endpoint Url.

6.2 Audit Bus

The broker presents the following interfaces in C++

6.2.1 Opening Sessions and Connections

```
int main(int argc, char** argv) {
    std::string broker = argc > 1 ? argv[1] : "localhost:5672";
    std::string address = argc > 2 ? argv[2] : "amq.topic";
    Connection connection(broker);
    try {
        connection.open();
        Session session = connection.createSession();
       // ### Your Code Here ###
        connection.close();
        return 0;
    } catch(const std::exception& error) {
```

```
        std::cerr << error.what() << std::endl;
        connection.close();
        return 1;
    }
}
```

6.2.2 Creating and Sending a Message

```
Sender sender = session.createSender(address);
sender.send(Message("Hello world!"));
```

6.2.3 Setting Message Content

```
Message message;
message.setContent("Hello world!");
// In some applications, you should also set the content type,
// which is a MIME type
message.setContentType("text/plain");
```

6.2.4 Receiving a Message

```
Receiver receiver = session.createReceiver(address);
Message message = receiver.fetch(Duration::SECOND * 1); // timeout
is optional
session.acknowledge(); // acknowledge message receipt
std::cout << message.getContent() << std::endl;
```

6.2.5 Receiving Messages from Multiple Sources

To receive messages from multiple sources, create a receiver for each source, and use session.nextReceiver().fetch() to fetch messages. session.nextReceiver() is guaranteed to return the receiver responsible for the first available message on the session.

```
Receiver receiver1 = session.createReceiver(address1);
Receiver receiver2 = session.createReceiver(address2);
Message message =  session.nextReceiver().fetch();
session.acknowledge(); // acknowledge message receipt
std::cout << message.getContent() << std::endl;
```

6.2.6 Replying to a message:

```
// Server creates a service queue and waits for messages
// If it gets a request, it sends a response to the reply to address
Receiver receiver = session.createReceiver("service_queue; {create:
always}");
Message request = receiver.fetch();
const Address& address = request.getReplyTo(); // Get "reply-to"
from request ...
if (address) {
  Sender sender = session.createSender(address); // ... send
response to "reply-to"
```

```
   Message response("pong!");
   sender.send(response);
   session.acknowledge();
 }
 // Client creates a private response queue - the # gets converted
 // to a unique string for the response queue name. Client uses the
 // name of this queue as its reply-to.
 Sender sender = session.createSender("service_queue");
 Address responseQueue("#response-queue; {create:always,
delete:always}");
 Receiver receiver = session.createReceiver(responseQueue);
 Message request;
 request.setReplyTo(responseQueue);
 request.setContent("ping");
 sender.send(request);
 Message response = receiver.fetch();
 std::cout << request.getContent() << " -> " << response.getContent()
<< std::endl;
```

6.2.7 Getting and Setting Standard Message Properties

This shows some of the most commonly used message properties, it is not complete.

```
 Message message("Hello world!");
 message.setContentType("text/plain");
 message.setSubject("greeting");
 message.setReplyTo("response-queue");
 message.setTtl(100); // milliseconds
 message.setDurable(1);
 std::cout << "Content: " << message.getContent() << std::endl
           << "Content Type: " << message.getContentType()
           << "Subject: " << message.getSubject()
         << "ReplyTo: " << message.getReplyTo()
         << "Time To Live (in milliseconds) " << message.getTtl()
         << "Durability: " << message.getDurable();
```

6.2.8 Getting and Setting Application-Defined Message Properties

```
 std::string name = "weekday";
 std::string value = "Thursday";
 message.getProperties()[name] = value;
 std:string s = message.getProperties()["weekday"];
```

6.2.9 Transparent Failover

If a connection opened using the reconnect option, it will transparently reconnect if the connection is lost.

```
 Connection connection(broker);
 connection.setOption("reconnect", true);
 try {
```

```
connection.open();
....
```

6.2.10 Maps

Maps provide a simple way to exchange binary data portably, across languages and platforms. Maps can contain simple types, lists, or maps.

```
// Sender
Variant::Map content;
content["id"] = 987654321;
content["name"] = "Widget";
content["probability"] = 0.43;
Variant::List colours;
colours.push_back(Variant("red"));
colours.push_back(Variant("green"));
colours.push_back(Variant("white"));
content["colours"] = colours;
content["uuid"] = Uuid(true);
Message message;
encode(content, message);
sender.send(message);

// Receiver
Variant::Map content;
decode(receiver.fetch(), content);
```

6.2.11 Guaranteed Delivery

If a queue is durable, the queue survives a messaging broker crash, as well as any durable messages that have been placed on the queue. These messages will be delivered when the messaging broker is restarted. Delivery is not guaranteed unless both the message and the queue are durable.

```
Sender sender = session.createSender("durable-queue");
Message message("Hello world!");
message.setDurable(1);
sender.send(Message("Hello world!"));
```

6.2.12 Transactions

Transactions cover enqueues and dequeues.

When sending messages, a transaction tracks enqueues without actually delivering the messages, a commit places messages on their queues, and a rollback discards the enqueues.

When receiving messages, a transaction tracks dequeues without actually removing acknowledged messages, a commit removes all acknowledged messages, and a rollback discards acknowledgements. A rollback does not release the message, it must be explicitly released to return it to the queue.

```
Connection connection(broker);
Session session =  connection.createTransactionalSession();
...
if (looksOk)
   session.commit();
else
   session.rollback();
```

6.2.13 Logging

The Qpidd broker and C++ clients can both use environment variables to enable logging. Use QPID_LOG_ENABLE to set the level of logging you are interested in (trace, debug, info, notice, warning, error, or critical):

```
export QPID_LOG_ENABLE="warning+"
```

Use QPID_LOG_OUTPUT to determine where logging output should be sent. This is either a file name or the special values stderr, stdout, or syslog:

```
export QPID_LOG_TO_FILE="/tmp/myclient.out"
```

7 License Information

University of Koblenz-Landau

All software produced in TAS3 by the IWM[10] at the University of Koblenz-Landau is released under the BSD license.

University of Nottingham

The Audit Bus is released under the TAS3 GPL license and the Apache QPID under the Apache license.

[10] Institut fuer Wissensmedien – Knowledge Media Institute

8 Roadmap and Conclusions

8.1 Audit Service

Concerning scalability the Audit Service has to be improved in further versions. We plan to set up several services that work together and are able to handle heavy loads of traffic. Moreover, we plan to provide other interfaces than SOAP. REST is not supported right now, but will be in the next version.

Furthermore, we have to adapt our Audit Service to newer versions of TAS3 Security Services like T3-PDP and T3-IDP.

8.2 Audit Bus

The QPID broker provides a method of reliable event data exchange in the TAS3 network. The AMQP design has been specifically chosen for its scalability which is something to test in the future. Future releases will improve the way that the Access Control is managed in the broker making the list more dynamic. The key challenge is to integrate the broker with the required components and ensure that its deployment in the trust network is able to support a flexible membership of services and users.

9 References

[1] TAS³ Deliverable D7.1 "Design of Identity Management, Authentication and Authorization Infrastructure", Version 3.0.1, 20 Dec 2010

[2] TAS³ Deliverable D2.1 "TAS³ Architecture", Version 2.0., June 2010

[3] http://www.amqp.org

[4] TAS³ Clients, TAS³ Deliverable D8.3, Version 2.0 December 2010

[5] TAS³ Deliverable D12.2 "Trust & Application Integration Report ", Version 1.7., July 2010

10 Appendix

Event Channels & TOPIC

1Session Events Channel: TOPIC = SES
a. Session creation (possibly even an anonymous session)
b. Session upgrade (e.g. SSO on an anonymous session, step-up auth)
c. Session refresh
d. Session termination
e. Session expiry
f. Session revival (if appropriate, could be used as a factor in authentication)

2 User Authentication Events Channel: TOPIC = UAEC
a. Positive
b. Failure with Retry
c. Definitive Failure

3 Token Issuing Channel: TOPIC= TIC
a. Tokens issued with:
i. Issuer
ii. Subject
iii. Audience
iv. Policy constraints
v. Validity time and/or usage count
vi. General content of the token
b. Token validation at relying party
c. Token use, to the appropriate extent
d. Token revocation when applicable

4 Authorization Channel: TOPIC = AC
a. Az request parameters
b. Az decision returned

5 Service Requester Channel:┬á TOPIC = SRQC
a. Choice of Service Provider
i. Discovery
ii. Hardwired choice of Service
iii. Automated or algorithmic Choice of Service
iv. Choice of Service solicited from the User
b. Trust negotiation steps
c. Consent to send data
d. Service Call event
i. Signature preparation, including choice of signing key

ii. Log of content of the message
iii. Peer authentication
iv. Success or failure to send message
e. Service Call exception
i. Redirect or end point change
ii. Recredentialing
iii. Interaction requested
iv. Replay after interaction
v. Dry-run
f. Service Call Response
i. Log of content of the message
ii. Peer authentication (usually by Request-Response pattern)
iii. Success or failure to receive message
g. Service Call Response exception
i. Failures, as detailed on the Faults Channel
ii. Application layer success or failure
h. Obligations processing
i. Presence of obligation
ii. Specific processing steps
iii. Failure to process obligation

6 Service Responder Channel: TOPIC = SRPC
a. Trust establishment and trust negotiation steps
b. Request Acceptance
c. Response filtering and authorization decision
d. Attachment of obligations

7 PII Collection Channel: TOPIC = PIICC

8 PII Release Channel; TOPIC = PIIRC

9 User Registration Channel: TOPIC = URC
a. Register
b. Modify
c. Deregister

10 SP Registration Channel: TOPIC = SPRC
a. Register
b. Modify
c. Change of Control
d. Deregister

11 User Reputation Channel: TOPIC = URC2
a. Explicit complaint or praise
b. Other events that affect reputation

12 Service Reputation Channel: TOPIC = SRC
a. Explicit complaint or praise
b. Other events that affect reputation

13 Browsing Event Channel (usually not shared) : TOPIC = BEC

14 Faults Channel: TOPIC = FC
a. Malformed protocol message
b. Insufficient sec mech
c. Signature verification fault
i. Malformed
ii. Crypto (public key or hash)
iii. Certificate validity (missing CA trust chain)
d. Inappropriate use
i. Audience
ii. Constraints
e. Expired tokens
f. Replay of message or token
g. Unsolicited message
h. Missing database entry
i. Explicit fault report

15 DoS Channel: TOPIC = DC
a. Invocation frequency alert
b. Data volume alert
c. Explicit DoS report (e.g. from monitoring organizations)

16 Intrusion Detection System and Firewall ACL Channel: TOP = IDFC
a. Scan alert
b. Attack fingerprint alert
c. Firewall deny rule triggered

17 Operations monitoring Channel: TOPIC = OMC
a. Server / Service
i. Up
ii. Down
iii. Scheduled downtime
iv. Congested
v. Retry
vi. Fail Over

18 Audit Operation Channel (very restricted circulation): TOPIC = AOC
a. Undertaking audits
b. Outcomes of audit

19 Billing Event Channel; TOPIC = BEVC

Amendment History

Ver	Date	Author	Description/Comments
2.00	01.12.10	M. Santos	First draft of deliverable
2.01	23.12.10	M. Santos	Modifications after internal review