OPAALS PROJECT

Contract n° IST-034824

WP5: Integration with the Digital Ecosystem Platform

Del 5.5 – Full integration of accountability, identity and trust and autopoiesis into D5.4

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D5.5

Due date: 31/12/2009

Delivery Date: September 2010

Short Description:

This document presents the integration work of Distributed Identity and Trust into the FlyPeer infrastructure. The work is representative of a software deliverable and as such much of the effort in the production of the software is represented in the production of open source projects to host the underlying work. Both distributed identity and distributed trust have been published as open source projects on SourceForge as IdentityFlow and TrustFlow respectively. While these works are standalone software components sets, this report shows how these have been integrated into the OPAALS FlyPeer infrastructure.

Author: WIT

Partners contributed: WIT

Made available to: All

Versioning		
Version	Date	Name, organization
0.1	**01/08/10**	**Paul Malone, WIT**
1	**30/08/10**	**Paul Malone, WIT**

Quality check

Internal Reviewers: Paolo Dini

Dependencies:

Work Packages	WP5: Task 5.9 – P2P infrastructure Implementation WP3: Task 3.15 – Distributed Identity, Accountability and Trust
Partners	WIT, IPTI
Domains	Computer Science domain: P2P networks, interactions, long-running transactions, distributed systems and networks, distributed identity, distributed trust. Social science domain: context-dependent trust, identity. Natural Science domain:
Targets	Computer, social and natural science researchers, SMEs, business analysts. Computer science communities: distributed computing, identity and trust.

Table of Contents

1 Introduction

This document presents the integration work of Distributed Identity and Trust into the FlyPeer infrastructure. The work is representative of a software deliverable and as such much of the effort in the production of the software is represented in the production of open source projects to host the underlying work. Both distributed identity and distributed trust have been published as open source projects on SourceForge as IdentityFlow and TrustFlow respectively. While these works are standalone software components sets, this report shows how these have been integrated into the OPAALS FlyPeer infrastructure.

The text in this document is also available in deliverable D3.12. This has occurred as it was not apparent what the differentiation of the deliverables were. This deliverable was originally to be delivered in Phase II of the project but due to the lack of availability of the FlyPeer platform at that time, was delayed to Phase III. In parallel to this there was a simialr deliverable described in WP3 (D3.12) which represents the same effort.

2 Infrastructure integration, Distributed identity, Trust and Accountability

2.1 Identity model and Implementation

The final identity model is described in D3.9. The identity model implementation, developed as part of the IdentityFlow (reference IdentityFlow & demo) sourceforge.net project, is described in D3.11.

IdentityFlow can be found at http://sourceforge.net/projects/identityflow/, while Flypeer can be found at http://kenai.com/projects/flypeer.

2.2 Integration points between identity work and Flypeer

There are a number of points at which the infrastructure, i.e. Flypeer, requires access to identity and identity related functionality. On OPAALS, identity work has encompassed a large breadth of issues including, theoretical user identity, naming and naming schemes, authentication, (single) sign-on, authorisation, digital credentials and PKI schemes. The core of WIT's contribution is IdentityFlow, the OPAALS identity model implementation, which is a platform agnostic toolkit for identity claim and verification, facilitating sign-on and attribute claim verification. Many of the areas listed above are not addressed directly by IdentityFlow, since work on these issues does not further WIT's contribution towards advancing the state of the art, however, in order to integrate the innovative identity work into Flypeer so as to produce useful and workable identity on the platform, much additional work was required.

The following integration points were identified and worked on,

1. A username and password (and other data), captured by Flypeer, can be sent to some Identity Provider (IdP) for authentication. Authentication can be triggered from within Flypeer (usually when the node starts) and the authn data is sent to appropriate authentication modules that have been provided as an add-on to IdentityFlow.

FlypeerMain.java (snippet)

```
AuthnData authnData = new AuthnData(authnType, username, password);
new SimplePeer(authnData, new Notifier());
```

PeerStarter.java (snippet)

```
PeerGroup defaultFlypeerGroup = new PeerGroupHandler().connectTo(
          PeerGroupHandler.FLYPEER_DEFAULT_PEER_GROUP, peer, 5000);
```

PeerGroupHandler.java (snippet)

```
peer.getAuthenticator().credentialAuthenticate(
                peerGroupFound, peer.getAuthnData());
```

2. Once a user has been authenticated, a Credential is created that can be used throughout the Flypeer platform to verify an authenticated identity. This Credential object is a JXTA Credential, which is valid for JXTA-based platforms, of which

Flypeer is one. The Credential is a custom implementation that contains additional encryption and IdP metadata. (See Credential XML Snippet below.)

```
(The snippet below is called by the credentialAuthenticate() method
in above snippet.)
```

Authenticator.java (snippet)

```
LocalCoordinatorCredential credential =
(LocalCoordinatorCredential) membershipService.join(authenticator);
```

3. Flypeer must present Credentials when making secure connections to other peers so that the destination peer can authenticate the incoming connection immediately.

```
Credentials are sent as payload in incoming connections that must
be authenticated. Credentials are serialized into XML for
transmission, and reconstructed remotely, using, respectively,

CredentialUtil.encodeCredential(Credential credential)

CredentialUtil.decodeCredential(String credentialEncrypted,
PeerGroup peerGroup)

(The validity of Credentials can be established as outlined in the
next section.)
```

4. An identity operation (see D3.9/3.11) must be launched by services to verify identity claims made by peers (that may or may not be authenticated at this stage). Peers must be able to determine whether the identity of other peers (using a different IdP) can be trusted. The following code snippet can be used by services to launch and identity operation.

SampleOperationLaunchTest.java (snippet)

```
/* Get the initial Connection. */
AssertionVerificationOperation operation =
AssertionVerificationOperation.assertionVerificationOperationInstan
ce();
JXTAConnection initialConnection =
(JXTAConnection)operation.getInitiatingConnection();
/* Local SAMLRequest into initialConnection dispatcher. */
JXTADispatcher dispatcher = initialConnection.getDispatcher();
dispatcher.setDestinationPeer(destinationPeer);
dispatcher.setPeerGroup(commsPeerGroup);
dispatcher.setChannel(Configuration.DEFAULT_OUTPUT_CHANNEL);
encoder.createAuthnRequest();
SAMLRequestParameterMap.LoaderMap samlRequest =
SAMLRequestParameterMap.loaderMapInstance(encoder.getAuthnRequest()
);
dispatcher.setAttributes(new SAMLRequestParameterMap(samlRequest));

/* Dispatch connection. */
initialConnection.dispatch();
```

2.3 Interactions between IdentityFlow and Flypeer

There are two main steps for verifying identity claims on the OPAALS DE. These are outlined below.

Step 1: Identifier Claims From Authenticating IdPs

JXTA provides a MembershipService framework for authenticating to PeerGroups, which IdentityFlow implements as an extension to its core functionality. This allows Flypeer to authenticate users to the main Flypeer group. The IdPMembershipService, which IdentityFlow implements, is available to each node on the JXTA network and operates as follows,

1. A set of login parameters, and the JXTA advertisement of a preferred IdP, is supplied via Flypeer as a JXTA Authenticator on the authenticating node.
2. The IdPMembershipService contacts the IdP node and supplies it with the correct authentication parameters.
3. The IdP returns an authentication result to the authenticating node, allowing the IdPMembershipService to complete it's PeerGroup 'join' attempt.
4. If authentication has been successful, the IdPMembershipService generates a credential from the result passed back from the IdP, which acts as proof that the entity's identity is indeed asserted by the IdP.
5. This credential can then be used as a means of filtering access to services in a JXTA-based environment.

The scheme for generating and using the credential assumes that all IdPs have a public-private key pair, and is as follows,

1. Following a successful authentication attempt, an IdP will sign the 'claimed identifier'6 with its private key.

2. The identifier, the IdP's public key, the encryption method and the signed identifier are packaged with other metadata into a JXTA Credential object.

3. Other entities can verify that the IdP in question authenticated the authenticating entity by obtaining the Credential ob ject and using the IdP's public key to decrypt the signed identifier and comparing it with the claimed identifier.

The Credential implementation used by IdentityFlow marshals and unmarshals to and from XML. An edited (for space) example of a marshaled Credential is given below.

```
<?xml version="1.0"?>

<!DOCTYPE jxta:Cred>

<jxta:Cred type="jxta:LocalCoordinatorCred" xml:space="..."
xmlns:jxta="http://jxta.org">

<PeerGroupID>urn:jxta:uuid-FEF..102<PeerGroupID>

<PeerID>urn:jxta:uuid-596..403<PeerID>

<SignedPeerID>MCwpf4i...K3O2oEPw=</SignedPeerID>
```

```
<SignAlgorithm>DSA</SignAlgorithm>

<GroupPublicKey>MICC...MYboJk=</GroupPublicKey>

<Username>identityflow@Guigoh</Username>

</jxta:Cred>
```

Step 2: Verifying an Identifier Claim

Identity claims are verified using identity operations, as discussed in sections 6 and 7. The identity operation specifies7

1. A set of contingent ordered connections that are executed in sequence (from one actor to the next), potentially with conditional logic affecting the protocol flow.
2. A set of actor tasks for each possible state of each actor in the protocol flow, that are executed when an actor assumes a particular state.
3. A binding, or set of bindings, that specify connection transport functionality in particular environments (e.g. JXTA binding).

A set of trust checks that must be made at appropriate points during protocol execution to ensure that there is sufficient trust between actors for the operation to succeed.

Identity operations are typically triggered by resource access, where it is necessary for entity identity claims to be accepted by the SP before the accessing entity can be granted access. Claims concerning identifiers issued by IdPs are important examples of claims that must be verified. In the particular case of the OPAALS DE, operations that verify user identifiers are simplified by the fact that each user propagates a credential which effectively encodes an identifier claim made by the user's IdP on behalf of the user. Therefore, only connections 1, 8 (service request and response) and 2, 7 (identifier claim verification request and response) in Figs. 5 and 6 need to be executed.

For all other identity claims, pertaining to any property that an entity can claim to possess and that its IdP can verify, identity operations are necessary.

Examples of other claims are "I am over the age of 18" or "I am a member of the OPAALS consortium" or "My real name is John Murphy". Whether or not these claims are accepted by other entities will depend on whether the accepting party's IdP trusts the claiming party's IdP in the context of asserting identity claims.

Since the OPAALS DE is a JXTA-based environment, a JXTA binding is used, which means that all operation connection messages (by current convention) are passed via JXTA pipes. Using JXTA transport functionality ensures entities operate in a pure P2P overlay, where entities can be contacted by name without the need for knowing a peer's underlying transport address (i.e. IP address); and also provides transparent firewall traversal.

Identity operations will tend to be designed to address a particular need, such as identifier verification, and will tend to be triggered by an actor seeking to verify a claim during the course of some activity. Therefore, it is anticipated the development of identity operations will be in response to the needs of SPs. However, it is likely that the template of the simple operation given in Fig. 5 will suffice for most purposes. An example of a more complicated identity claim would be a situation where an SP would not accept a particular claim unless two or more IdPs asserted the claim on behalf of the user. This would necessitate a more complicated protocol flow involving 5 rather than 4 actors. In general, if any of the items in

the list above change, modification will be required, which should be simplified by the modular, extensible design chosen by IdentityFlow.

2.4 Trust Model and Implementation

The final trust model is described in D3.9. The implementation, developed as part of the TrustFlow[1] sourceforge.net project, is described in D3.11.

The model is shown below in Figure 1. The approach is to use a trust overlay network for providing a community based approach to trustworthiness based on the reputation of entities. The Entity can represent a node, service, resource, a service provider or a service consumer. Each entity has a Trust Manager associated with it. The entities gain experience from interacting with other entities and publish reports of these experiences to the Trust Manager. Using a pre-defined context dependent algorithm, the Trust Manager updates the entities' local trust and based on a policy of sharing trust information provides trust updates to other Trust Managers in the overlay network.

[1] TrustFlow project, a set of Java components to enable a referral based trust and reputation system, http://sourceforge.net/projects/trustflow/

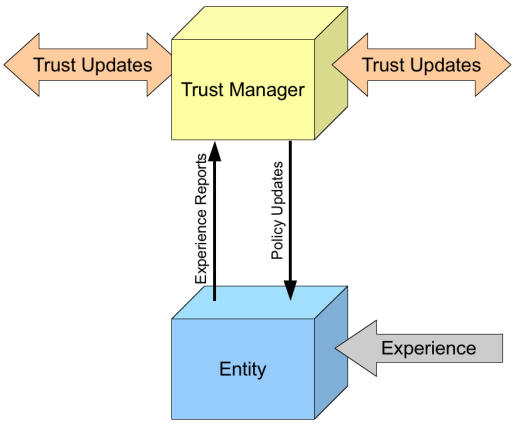

Figure 1: OPAALS Trust Overlay Model

2.5 ReferralHandler

The transfer of referrals is defined through the use of a `ReferralHandler` interface (shown in Figure 2). The interface defines two methods, `getReferral()` which is used request remote referrals via a `ReferralQuery()` and `putReferral()` which is used to make available or publish a `TrustValue` as a referral. This interface is then implemented for each platform that TrustFlow supports.

```
package org.opaals.trust;
public interface ReferralHandler {
	TrustValue getReferral(ReferralQuery query);
	void putReferral(String trustor, TrustValue tv);
}
```

Figure 2: ReferralHandler Interface

2.6 JXTA Implementation

In the case of FlyPeer, which is a JXTA based platform, an implementation called `JxtaReferralHandler` using the JXTA Resolver API was developed. The `QueryHandler` interface provides a means in JXTA to query and receive messages via a query response messaging paradigm. `QueryHandler` defines two methods for this, `processQuery()` for processing received queries and `processResponse()` for processing responses to queries previously issued. The `ResolverQueryMsg` and `ResolverResponseMsg` classes allow for associating responses with queries. See Figure 3 for details.

```
package org.opaals.trust.jxta.impl;
public class JxtaReferralHandler implements ReferralHandler, QueryHandler {
        public TrustValue getReferral(ReferralQuery referralQuery) {
        // implement creating a query, send it and return response if available
        }
        public void putReferral(String trustor, TrustValue tv) {
        // implement making a TrustValue available for transferral
        }
        public int processQuery(ResolverQueryMsg query) {
        // receive a query and process it
        }
        public void processResponse(ResolverResponseMsg msg) {
        // receive a response to a query and transfer to TrustValue
        }
}
```

Figure 3: JXTAReferralHandler class

In addition to implementing the methods of the `ReferralHandler` and `QueryHandler` the `JxtaReferralHandler` class provides a constructor which takes a JXTA `PeerGroup` object as a parameter in order to associate the queries with a `ResolverService` for that `PeerGroup`.

2.7 Integrating TrustFlow with FlyPeer

As described in D3.11, the `TrustManager` is the core of TrustFlow and has the following responsibilities:

1. Maintaining a set of algorithms for connected `Trustor` entities for specific contexts. Each context can have distinct algorithms for determining trust based on direct experience, referrals and reputation.

2. Receiving `ExperienceReports` from the `Trustor` entities and generating updated trust values according to the appropriate algorithms.

3. Providing updated `TrustValues` to other `TrustManagers` which can be used as inputs to referral or reputation algorithms. It is only appropriate that these published `TrustValues` have been derived from direct experience.

4. Providing `PolicyUpdates` to the `Trustor` entities based on current `TrustValues` derived from direct experience and referrals and reputation.

In order to integrate the facilities of TrustFlow into FlyPeer it is required that FlyPeer services and service consumers can access a `TrustManager` object and query it for trust values as well as publish trust values for referrals. This is initially modelled by adding two methods to the base `Peer` interface in FlyPeer to get and set the `TrustManager` object. This is shown in Figure 4.

```
package br.org.ipti.flypeer.model;

public interface Peer {

	...

	TrustManager getTrustManager();

	void setTrustManager(TrustManager trustManager);

}
```

Figure 4: TrustFlow Additions to the Peer Interface

These two methods are then implemented in in the abstract class `AbstractPeer` which maintains a private member object instance of `TrustManager`.

Each FlyPeer node is instantiated in a Thread through a class called `PeerStarter`. In this method the `TrustManager` is initialised using the `JxtaReferralHandler` and the `setTrustManager()` method is called on the Peer object. The initialisation of the `TrustManager` is shown below in Figure 5.

```
package br.org.ipti.flypeer.main;

public class PeerStarter implements Runnable{

	...

	private TrustManager initTrustManager(PeerGroup peerGroup){

		JxtaReferralHandler handler = new JxtaReferralHandler(peerGroup);
		TrustManager trustManager = new SimpleTrustManager();
		trustManager.setReferralHandler(handler);
		AlgorithmConfigurator algoConfig = new
			AlgorithmConfigurator(peer.getUsername(),
			trustManager,
			"algoConfigurationFile.xml");

		return trustManager;

    }
}
```

Figure 5: Snippet from PeerStarter showing TrustManager Initialisation

Once this initialisation is complete a service or service consumer can retrieve the `TrustManager` and assess trust ratings as described in deliverable D3.11.

2.8 Configuring Trust in FlyPeer

While TrustFlow is integrated into the FlyPeer platform, it might not be required for all cases. In fact the default behaviour of FlyPeer is not use these trust facilities. In order to enable TrustFlow for Service Providers and Consumers who require it, a configuration file called `trustFlow.properties` is used to indicate how to configure trust in the platform. A sample configuration is shown below in

```
activate=true
persistence=true
```

Figure 6: trustFlow.properties Sample configuration

2.9 Accountability

The Accountability model is described in deliverable D3.8 and the interfaces for implementation is described in D3.11. The model was published in [Mal09], where it won "Best Paper Award".

While a simulation of the model using the interfaces defined in D3.11 has been performed successfully in-house, a full blown implementation integrated with FlyPeer was not performed due to a lack of resources available within the lifetime of the project.

3 References

[Mal09] Malone, P., Jennings, B., *Distributed support for public and private accountability in digital ecosystems,* in proceedings of MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems, pp 391-398, 2009, ACM, available at http://doi.acm.org/10.1145/1643823.1643895