SemEUsE

SEMantiquE pour bUS de sErvice

2007 TLOG 018

Document name:	SemEUsE QoS and dependability support framework
Document version:	0.2
Task code:	T2.2
Deliverable code:	T2.2D2
WP Leader (organisation):	EBM
Deliverable Leader (organisation):	EBM
Authors (organisations):	EBM, FT
Date of first version:	17/07/2009

Change control

Changes	Author / Entity	Code of version
Creation of the document - 17/07/09	EBM	V0.1
Add ebm part	EBM	V0.2
Update chapter 5	FT	V0.3

Table of Contents

1.

2. Introduction

In deliverable T2.2D1, we have defined necessary concepts allowing us to insert domain and technical QoS into the description of the services. The addition of these non-functional properties will allow us to describe complex SLAs and to establish a global policy in terms of performance, dependability and security on all the composition of services. Non-functional properties of services can be expressed in many different ways. These non-functional properties are commonly referred to as Quality of Service properties, or QoS.

In SEMEUSE, we are particularly interested by QoS ontology, where semantically described QoS properties complement semantic functional service descriptions and are considered in service discovery and composition as well as in service execution with Service Level Agreements (SLA).

Assurance of such properties implies special design features and related runtime mechanisms for services, e.g. transactional features of services or underlying middleware support such as transparent service replication, where the latter calls for customized ESB middleware connectors. The aim of this deliverable is to define a framework able to support and handle QoS ontology and semantic concept to achieve SLA enforcement and QoS of services.

This document is organized as follows. Section 2 presents PEtALS, the support for Dependability and QoS of services. The monitoring service is explained in Section 3. The management of SLA contracts is achieved by the SLO Manager presented in Section 4. The security concerns with the Single-Sign-On Service are detailed in Section 5. Section 6 concludes this deliverable.

These works will be achieved by EBM WS, INT, THALES, LIP6, INSA, FT and INRIA ARLES.

3. Semeuse: a Support Framework for Dependability and QoS of Services

This chapter presents the Semeuse support framework and available modules able to achieve some security and dependability concerns in PEtALS.

3.1. Semeuse Architecture

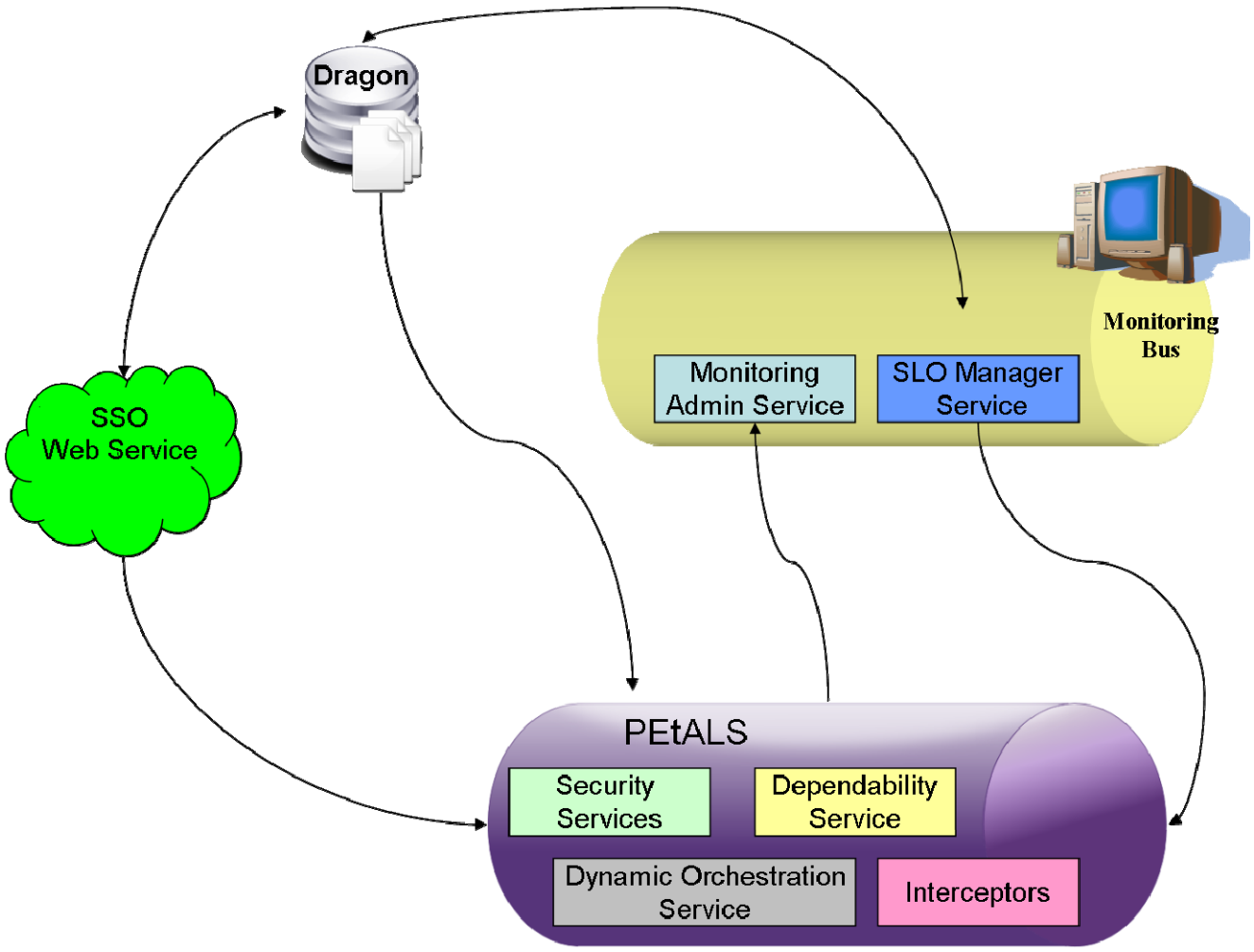


Figure 31: Semeuse Architecture

Figure 31 presents the Semeuse architecture. This framework is composed of 4 components:

- PEtALS: the enterprise Service Bus. It contains several services such as Security Services, Dependability Service, …etc.
- Dragon: the enterprise governance and registry of services.
- The monitoring bus able to monitor business services on PEtALS and verify SLA contract.
- The Single-Sign-ON (SSO) service is mechanism whereby a single action of user authentication and authorization can permit a user to access all computers and systems where he has access permission

3.2. PEtALS Entreprise Service Bus

At the cornerstone of the information system, Petals provides a natively distributed architecture which enables an agile and interoperable approach for an efficient management of information flows across extended organisations. Built on standard and open technologies, Petals supports all the non-functionnal aspects of service exposition outside the application domain such as security and dependability concerns.

3.2.1. Security Services organisation in PEtALS

Discussion sur les différents niveaux (transport and co). Expliquer comment sécuriser de bout en bout

- ⇨ INSA : security actuelle du bus => transporteur Dream securisé; signature, JAAS… etc.

3.2.2. Dependability service in PEtALS

In PEtALS, Service consumers may identify needed services by WSDL interface name, not by end point address. This decouples the consumer from the provider, and allows the NMR (Normalized Message Router, the internal PEtALS router) to select appropriate providers.

To select appropriate providers, several strategies can be configured when services are stateless.

Primary/Backup strategy

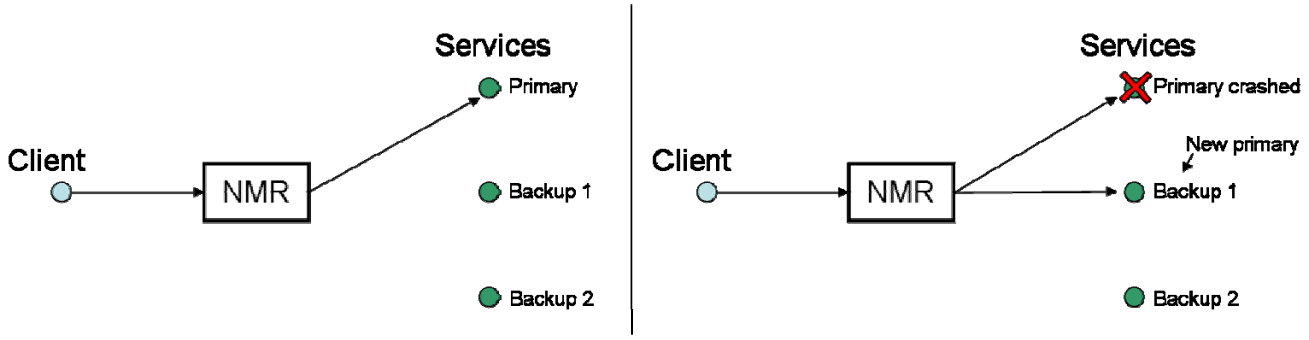


Figure 32: Primary/Backup Strategy

In primary/backup strategy, the NMR chooses the first service (the primary) corresponding to interface wanted by the client. If the primary responds, this response is sent by the NMR to the client. If the primary failed, the NMR chooses the next backup (it becomes the new primary) and send the request.

Leader/Follower Strategy

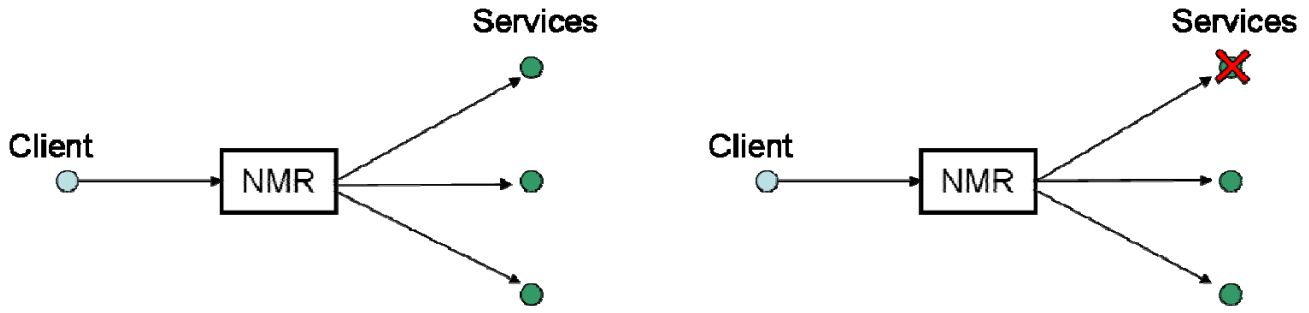


Figure 33: Leader/Follower Strategy

In leader/Follower strategy, the NMR sends the request to all providers, the first response is sent to the client.

Vote Strategy

Like the leader/Follower strategy, in the vote strategy, the NMR sends the request to all providers but collects all responses to compare them before sending the valid response to the client.

3.3. Dragon Entreprise Governance

Dragon gives you day-to-day control over your service infrastructure such as PEtALS. It allows service and user management, SLA definition, provides service registry, triggers alerts, and defines rules among users. It becomes a valuable tool, once your infrastructure goes in production, and services are deployed, to handle the rush of service creation.

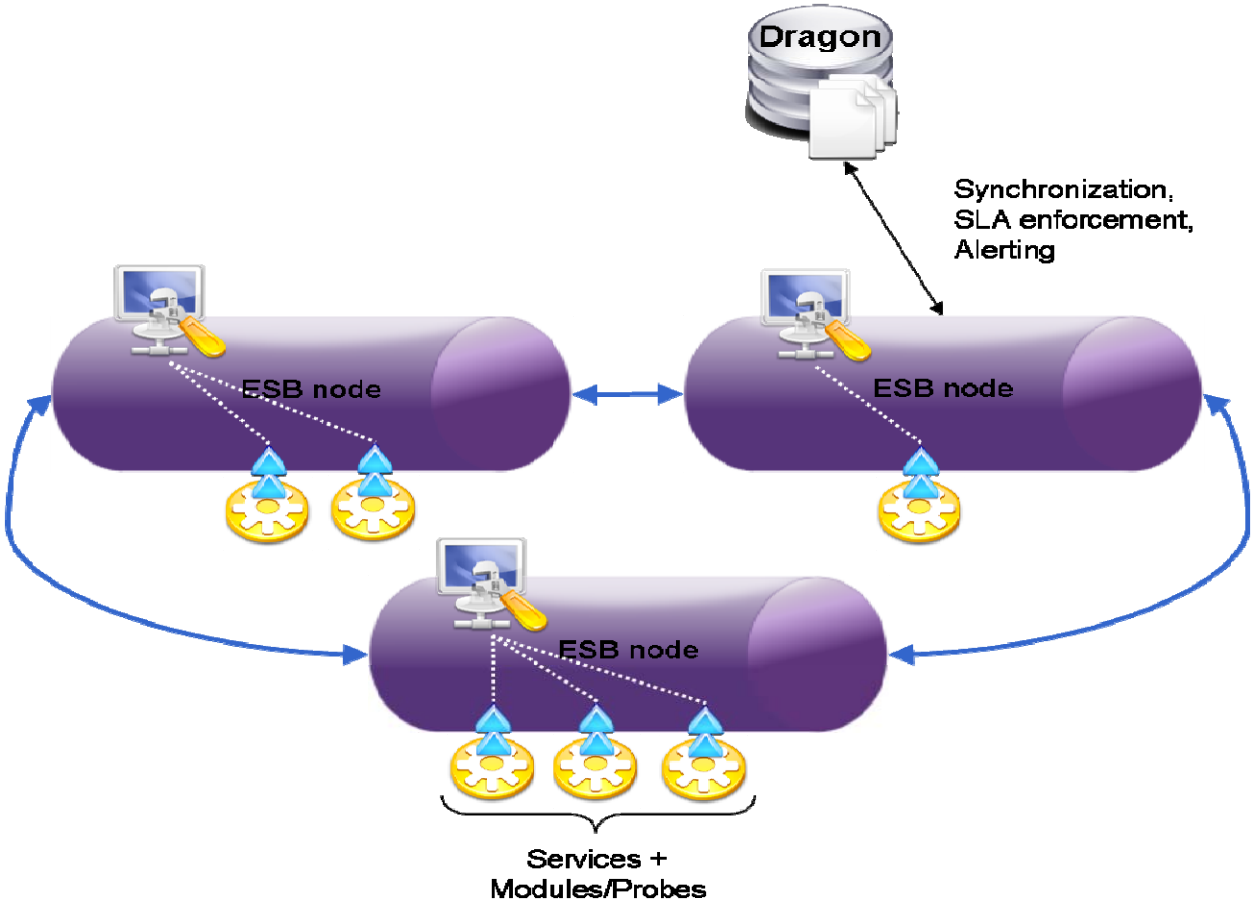


Figure 34: Interactions between PEtALS and Dragon

Figure 1 presents interactions between Dragon and PetALS. When it connected to a PetALS node, Dragon is able to get all service's descriptions of all nodes in the domain. The next sections will present how Dragon and PETALS are able to monitore, secure and achieve SLA enforcement to improve QoS of Services.

3.4. The monitoring bus

The monitoring bus is in charge to monitor and throw alert when SLO (Service Level Objective) is not respected between a consumer and a provider. The monitoring bus contains these following services:

- Monitoring Admin Service explained in Section 3.
- SLO Manager service explained in Section 4.

This monitoring bus controls also the acitivty of PEtALS and so, it is external to PEtALS.

3.5. Security services

In order to improve dependability, the security service is designed as an external part, able to be connected either to the ESB or to the governance environment. It implements the necessary toolset to support security patterns organised to support security requirements:

- Confidentiality: this involves providing access control features as well as a secured transport

- Integrity: this involves signing and authenticating messages. This feature also support the non repudiation service as messages are signed by the service which has sent it.

Security requirements are expressed either by the service provider (while defining semantic security annotation and building QoP agreement (see T2.1D2 and T2.2D1) and by the service customer. In order to simplify the security policy description security patterns are built as answers to security objectives defined thanks to a simplified ontology (**Erreur ! Source du renvoi introuvable.**).

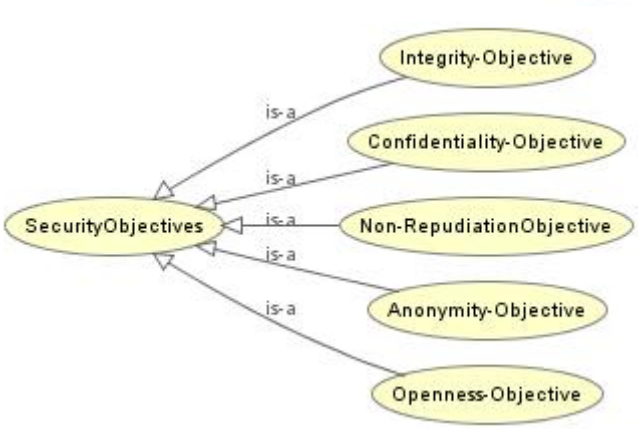


Figure 35: Security objectives ontology

These patterns are then used to set an adapted XML parameter file which will be used by the security mediator to coordinate and compose the convenient security services (**Erreur ! Source du renvoi introuvable.**).

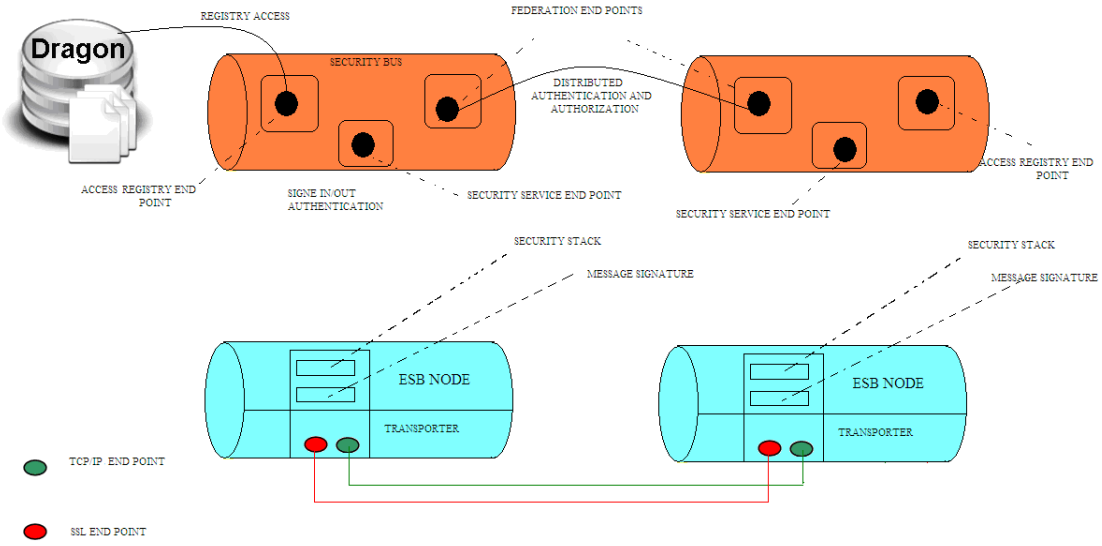


Figure 36: Security service organisation

4. Monitoring Service

4.1. WSDM Specification

The Web Services Distributed Management (WSDM) specification for the Management Of Web Services (MOWS) **[MOWS]** provides a means of managing Web services.

MOWS is an extension and application of the WSDM Management Using Web Services (MUWS) **[MUWS]** specification. MOWS is an extension of MUWS in that MOWS uses and extends the capabilities defined in MUWS. At is simplist level, MOWS uses the same resource properties exposed by MUWS manageability capabilities. For example, the MOWS OperationalStatus capability specifies the same resource property as defined in MUWS. MOWS extends other MUWS manageability capabilities in much the same way as UML classes extend their parent classes. New properties that are special to Web services are added to those properties already defined for MUWS capabilities. For example, the MOWS Metrics capability extends the MUWS Metrics capability with metrics that are appropriate for Web services.

The focus of MOWS is on Web services as WSDM manageable resources. MOWS defines capabilities, which includes properties, operations, and events, for addressing the monitoring requirements of Web services. In so doing, it both uses MUWS as an application framework as well as extending several MUWS capabilities.

In fact, WSDM-MOWS allows us to define a non-functional service endpoint able to monitor a functional service endpoint.

This non-functional WSDM endpoint is a producer of notification based on WS-Notification standard. This producer is able to support the topics defined in the xml file below:

```
<?xml version="1.0" encoding="utf-8"?>
<wstop:TopicNamespace name="MOWS"
    targetNamespace="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows-events.xml"
    xmlns:muws-xs2="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part2.xsd"
    xmlns:muws-xs1="http://docs.oasis-open.org/wsdm/2004/12/muws/wsdm-muws-part1.xsd"
    xmlns:mows-xs="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows.xsd"
    xmlns:wstop="http://docs.oasis-open.org/wsn/t-1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/wsn/t-1 http://docs.oasis-open.org/wsn/t-1.xsd">

    <wstop:Topic name="IdentificationCapability" messageTypes="muws-xs1:ManagementEvent" />
    <wstop:Topic name="MetricsCapability" messageTypes="muws-xs1:ManagementEvent" />
    <wstop:Topic name="OperationalStateCapability"  messageTypes="muws-xs1:ManagementEvent" />
    <wstop:Topic name="OperationalStatusCapability" messageTypes="muws-xs1:ManagementEvent" />
    <wstop:Topic name="RequestProcessingStateCapability" messageTypes="muws-xs1:ManagementEvent" />
    <wstop:Topic name="RequestProcessingObservations" messageTypes="muws-xs1:ManagementEvent">
        <wstop:Topic name="RequestReceived" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestProcessing" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestCompleted" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestFailed" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="Digest" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
    </wstop:Topic>

    <wstop:Topic name="RequestProcessingObservationsWithAttachments" messageTypes="muws-xs1:ManagementEvent">
        <wstop:Topic name="RequestReceived" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestProcessing" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestCompleted" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="RequestFailed" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
        <wstop:Topic name="Digest" messageTypes="muws-xs1:ManagementEvent"></wstop:Topic>
    </wstop:Topic>
</wstop:TopicNamespace>
```

Figure 47: WSDM Topics

Among these topics, the “MetricCapability” topic is really interesting in Semeuse Project. This topic allows client to known the QoS of each operation of a service. When a client subscribes to this topic on a producer, it can

be notified by messages containing OperationMetrics xml element. OperationMetrics is presented in **Erreur ! Source du renvoi introuvable.**.

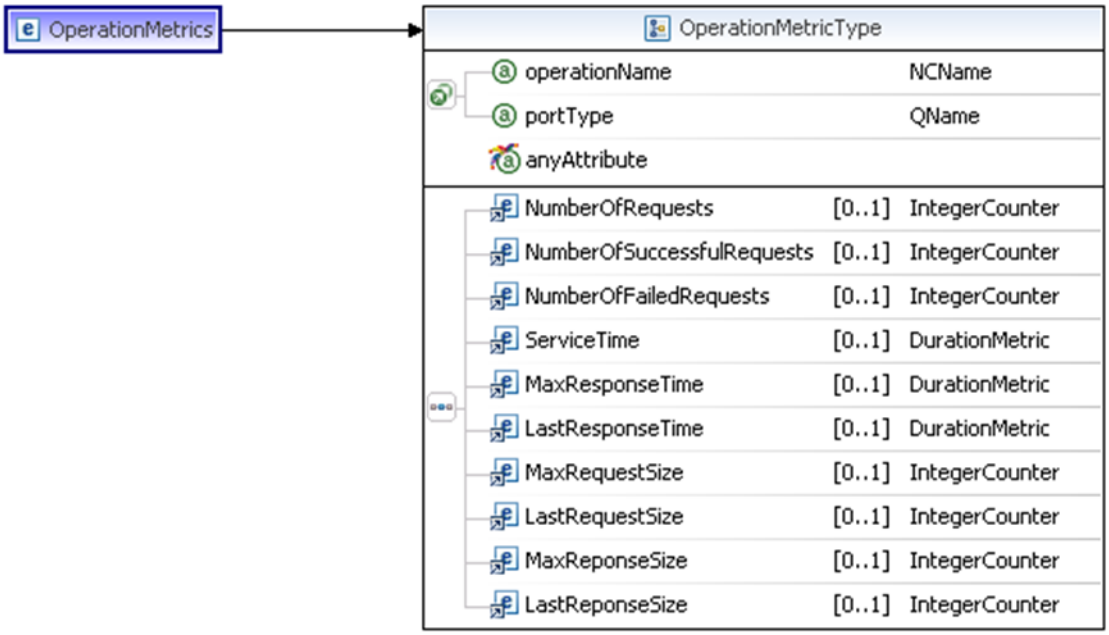


Figure 48: WSDM Operation Metrics

4.2. Monitoring Service

The **Erreur ! Source du renvoi introuvable.** presents the generic WSDM monitoring service interface for any monitoring endpoints. This interface is composed of these following operations:

- GetResourceProperty: this method allows us to known all supported topics by the producer service.
- Subsribe: this method allows a client to subcribe to one topic of provider.
- Unsubsribe: this method allows a client to unsubcribe to one topic of provider.
- GetCurrentMessage: this method allows a client to get the current message of a topic given in parameters.

WSDMMonitoringServiceInterface		
GetResourceProperty		
input	GetResourcePropertyRequest	GetResourceProperty
output	GetResourcePropertyResponse	GetResourcePropertyResponse
ResourceUnknownFault	ResourceUnknownFault	ResourceUnknownFault
InvalidResourcePropertyQNameFault	InvalidResourcePropertyQNameFault	InvalidResourcePropertyQNameFault
Subscribe		
input	SubscribeRequest	Subscribe
output	SubscribeResponse	SubscribeResponse
ResourceUnknownFault	ResourceUnknownFault	ResourceUnknownFault
InvalidFilterFault	InvalidFilterFault	InvalidFilterFault
TopicExpressionDialectUnknownFault	TopicExpressionDialectUnknownFault	TopicExpressionDialectUnknownFault
InvalidTopicExpressionFault	InvalidTopicExpressionFault	InvalidTopicExpressionFault
TopicNotSupportedFault	TopicNotSupportedFault	TopicNotSupportedFault
InvalidProducerPropertiesExpressionFault	InvalidProducerPropertiesExpressionFault	InvalidProducerPropertiesExpressionFault
InvalidMessageContentExpressionFault	InvalidMessageContentExpressionFault	InvalidMessageContentExpressionFault
UnacceptableInitialTerminationTimeFault	UnacceptableInitialTerminationTimeFault	UnacceptableInitialTerminationTimeFault
UnrecognizedPolicyRequestFault	UnrecognizedPolicyRequestFault	UnrecognizedPolicyRequestFault
UnsupportedPolicyRequestFault	UnsupportedPolicyRequestFault	UnsupportedPolicyRequestFault
NotifyMessageNotSupportedFault	NotifyMessageNotSupportedFault	NotifyMessageNotSupportedFault
SubscribeCreationFailedFault	SubscribeCreationFailedFault	SubscribeCreationFailedFault
GetCurrentMessage		
input	GetCurrentMessageRequest	GetCurrentMessage
output	GetCurrentMessageResponse	GetCurrentMessageResponse
ResourceUnknownFault	ResourceUnknownFault	ResourceUnknownFault
TopicExpressionDialectUnknownFault	TopicExpressionDialectUnknownFault	TopicExpressionDialectUnknownFault
InvalidTopicExpressionFault	InvalidTopicExpressionFault	InvalidTopicExpressionFault
TopicNotSupportedFault	TopicNotSupportedFault	TopicNotSupportedFault
NoCurrentMessageOnTopicFault	NoCurrentMessageOnTopicFault	NoCurrentMessageOnTopicFault
MultipleTopicsSpecifiedFault	MultipleTopicsSpecifiedFault	MultipleTopicsSpecifiedFault
Unsubscribe		
input	UnsubscribeRequest	Unsubscribe
output	UnsubscribeResponse	UnsubscribeResponse
ResourceUnknownFault	ResourceUnknownFault	ResourceUnknownFault
UnableToDestroySubscriptionFault	UnableToDestroySubscriptionFault	UnableToDestroySubscriptionFault

Figure 49: WSDL Interface of a WSDM Monitoring Service

The notification methods such as Subscribe and Unsubscribe are really important in this context. It allows client a receive notification message on the QoS of functional endpoint in an asynchronous way.

In Semeuse project, this monitoring service is used by several clients:

- Dragon (eBM): to get QoS property of functional service in the aim to allow users to choose their appropriate service.
- Monitoring Console (Tuvalu): to allows user to observe the state of functional services at runtime.
- SLO Manager (FT): able to trigger alerts when a violation of SLO is observed.

4.3. Interactions between Monitoring Service and PEtALS

The **Erreur ! Source du renvoi introuvable.** presents the main interaction between Monitoring Service and PEALS. The monitoring service is deployed on a bus: Monitoring Bus. It contains a Monitoring Admin Service able to create or destroy a monitoring service endpoint on the monitoring bus.

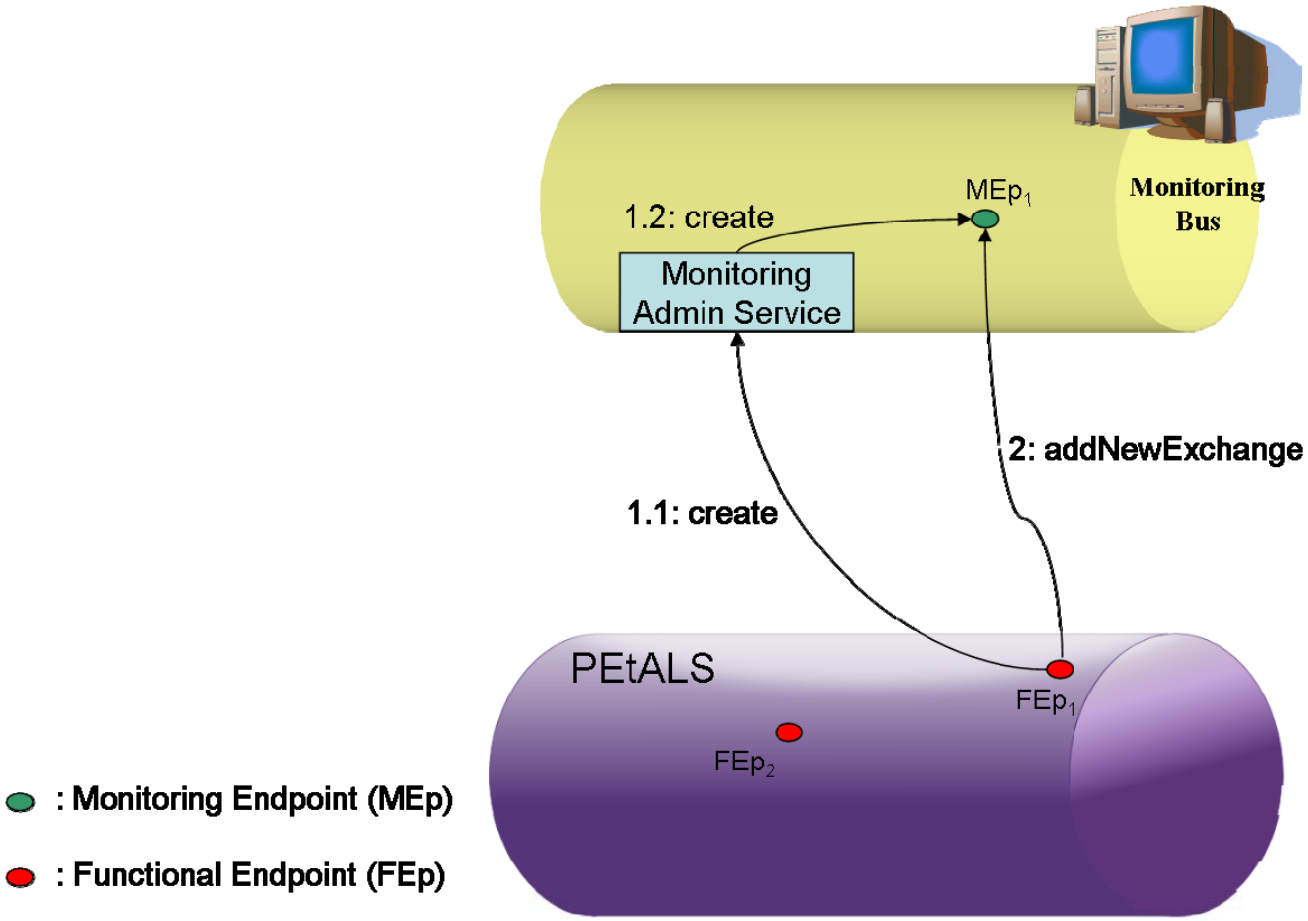


Figure 410: connection between Petals and Monitoring Service.

When a functional endpoint is deployed on PEtALS, and if it is configured for that, it can ask to Monitoring Admin Service (1.1 create on **Erreur ! Source du renvoi introuvable.**) to create his associated monitoring endpoint in monitoring bus (1.2 create on **Erreur ! Source du renvoi introuvable.**). The endpoint address of monitoring endpoint is returned in the response of the “create” operation.

So, when the functional endpoint receives a client request, it processes the request and sends a trace of this exchange to the associated monitoring endpoint using the “addNewExchange” operation. The signature of the “addNewEchange” operation can be seen below:

```
void addNewExchange(QName serviceName, String endpointName, String operationName,
        Document request, Document response,
        Date startExchange, Date endExchange, responseType type) throws MonitoringException;
```

All information concerning the exchange between the functional consumer and provider are sent to the monitoring endpoint. This operation is contained in a private interface accessible only by the functional endpoint.

From these data, the monitoring can deduce the QoS metrics of the concerned operation of service and notify this new metrics to its clients (like Dragon for instance).

4.4. Interactions between Monitoring Service and Dragon

Dragon uses monitoring bus like a classical client. It subscribes (3: Subscribe) on a monitoring endpoint and waits the notifications (4: Notify).

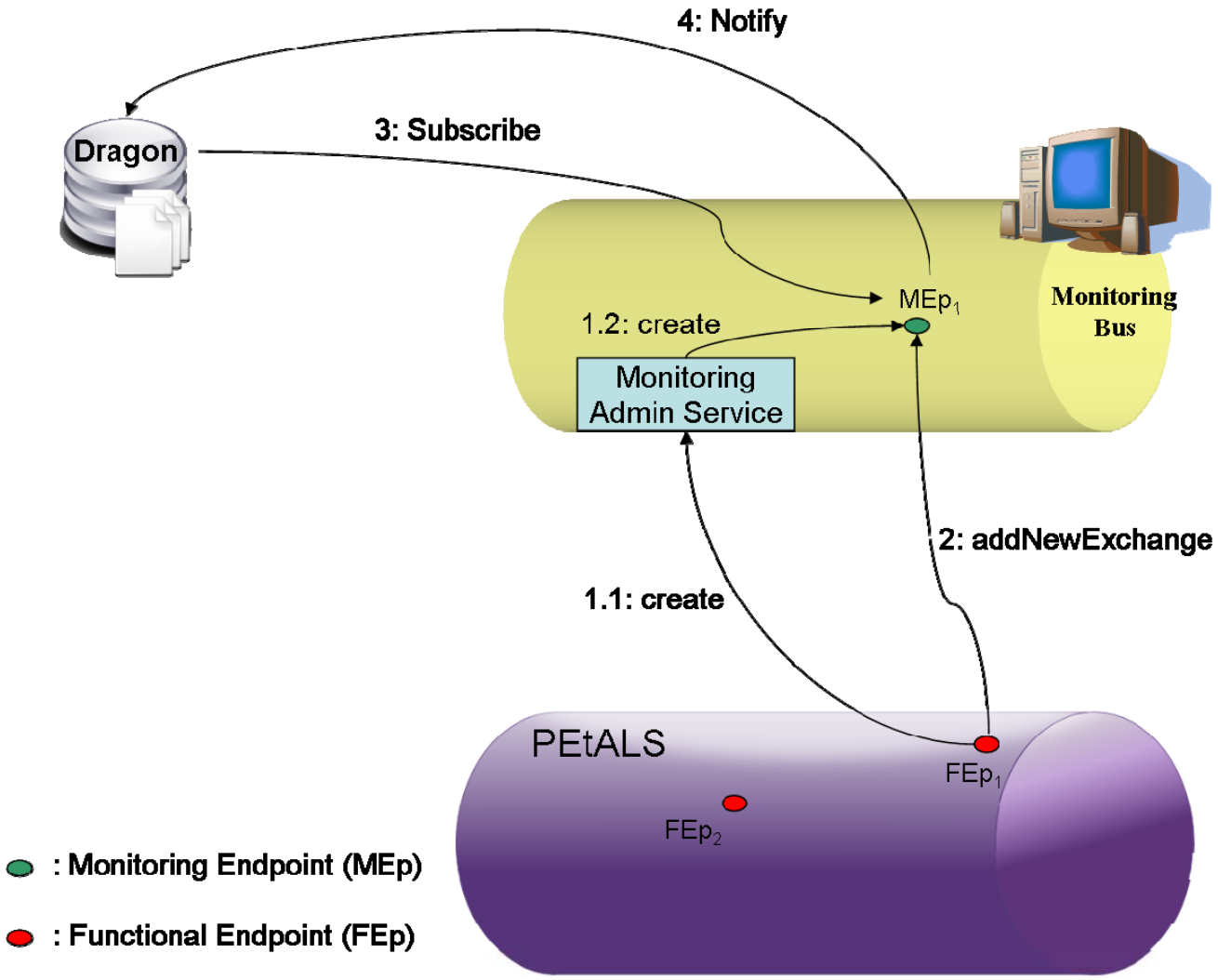


Figure 411: connection between Dragon and Monitoring Service

Figure 411 presents an example of notification sent by a monitoring endpoint. You can see that the "NotificationContent" element contains an "OperationMetrics" element concerning one operation of functional endpoint.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns6:Notify xmlns:ns6="http://docs.oasis-open.org/wsn/b-2"
    xmlns="http://www.w3.org/2005/08/addressing" xmlns:ns10="http://docs.oasis-open.org/wsrf/r-2"
    xmlns:ns2="http://www.ebmwebsourcing.com/mapping/WSNotifEndpointReference"
    xmlns:ns3="http://org.ow2.petals/ebmwebsourcing/specific/resourceIds"
    xmlns:ns4="http://docs.oasis-open.org/wsrf/bf-2" xmlns:ns5="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:ns7="http://docs.oasis-open.org/wsrf/rl-2" xmlns:ns8="http://docs.oasis-open.org/wsn/t-1"
    xmlns:ns9="http://docs.oasis-open.org/wsn/br-2">
   <ns6:NotificationMessage>
      <ns6:SubscriptionReference>
         <Address>{http://ftparis.com}ftParisEndpoint</Address>
         <ReferenceParameters>
            <ns2:ResourcesUuid>
               <ns2:Uuid></ns2:Uuid>
            </ns2:ResourcesUuid>
         </ReferenceParameters>
      </ns6:SubscriptionReference>
      <ns6:Topic xmlns:tns="http://docs.oasis-open.org/wsdm/2004/12/mows/wsdm-mows-events.xml"
          Dialect="http://docs.oasis-open.org/wsn/t-1/TopicExpression/Concrete">tns:MetricsCapability</ns6:Topic>
      <ns6:ProducerReference>
         <Address>{http://petals.ow2.org}stockquoteEndpointMonitoring</Address>
      </ns6:ProducerReference>
      <ns6:Message>
         <wsnebm:NotifyContent xmlns:wsnebm="http://www.ebmwebsourcing.com/WS-BaseNotification/NotifyContent">
&lt;ns4:resourceProperties xmlns:ns4="http://docs.oasis-open.org/wsdm/mows-2.xsd" xmlns="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://docs.oasis-open.org/wsdm/muws1-2.xsd" xmlns:ns3="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
xmlns:ns5="http://docs.oasis-open.org/wsdm/pbm.xsd" xmlns:ns6="http://www.webserviceX.NET/"
operationName="GetQuote" portType="ns6:StockQuoteSoap"&gt;
   &lt;ns4:NumberOfRequests ResetAt="2009-07-20T17:44:12.390+02:00"&gt;1&lt;/ns4:NumberOfRequests&gt;
   &lt;ns4:NumberOfSuccessfulRequests ResetAt="2009-07-20T17:44:12.390+02:00"&gt;1&lt;/ns4:NumberOfSuccessfulRequests&gt;
   &lt;ns4:NumberOfFailedRequests ResetAt="2009-07-20T17:44:12.390+02:00"&gt;0&lt;/ns4:NumberOfFailedRequests&gt;
   &lt;ns4:ServiceTime ResetAt="2009-07-20T17:44:12.390+02:00"&gt;PT0.000S&lt;/ns4:ServiceTime&gt;
   &lt;ns4:MaxResponseTime/&gt;
   &lt;ns4:LastResponseTime LastUpdated="2009-07-20T17:44:12.390+02:00" &gt;PT0.000S&lt;/ns4:LastResponseTime&gt;
   &lt;ns4:MaxRequestSize/&gt;
   &lt;ns4:LastRequestSize LastUpdated="2009-07-20T17:44:12.359+02:00" &gt;125&lt;/ns4:LastRequestSize&gt;
   &lt;ns4:MaxReponseSize/&gt;
   &lt;ns4:LastReponseSize LastUpdated="2009-07-20T17:44:12.390+02:00" &gt;125&lt;/ns4:LastReponseSize&gt;
&lt;/ns4:resourceProperties&gt;
         </wsnebm:NotifyContent>
      </ns6:Message>
   </ns6:NotificationMessage>
</ns6:Notify>
```

Figure 412: Example of notification sent by monitoring endpoint producer

5. QoS and SLA Checking Service

This chapter gives an overview of the architecture of the Service Level Checking (SLC) for SLAs, which has been introduced in PEtALS. The SLC is a service of the Monitoring Bus. SLC is in charge of checking if the agreements (i.e. the SLAs) defined between the clients and the service providers are respected or violated (as a side note, the SLAs are specified via Dragon: http://dragon.ow2.org). SLC's results are displayed via an IHM designed and developed by INRIA Tuvalu. Detailed information regarding the SLC itself can be found in the deliverable T3.1D2 (accessible via http://www.semeuse.org).

The proposed SLC for SLAs is based on an innovative architecture for data collection. The data collection provided is well in line with the data mediation concept [Wie92] [GMPQ+97] [Ric09]. Using this architecture enables the isolation of several preoccupations:

- Indicators collection and synchronization.
- Indicators processing.
- Forwarding processed indicators to the SLC service via the needed protocol and format.

The innovative architecture proposed is a multi-layered architecture. Three main layers were identified:

- The Core monitoring layer, which is the lowest layer. It is in charge of collecting low-level indicators from PEtALS.
- The Data collector layer. This is the intermediate layer. It is in charge of processing the low-level indicators into high-level business-oriented indicators.
- And, finally, the Service monitoring layer, which is the highest layer. This last layer is in charge of the Service Level Checking. To do so, it uses the high-level indicators produced by the Data collector layer.

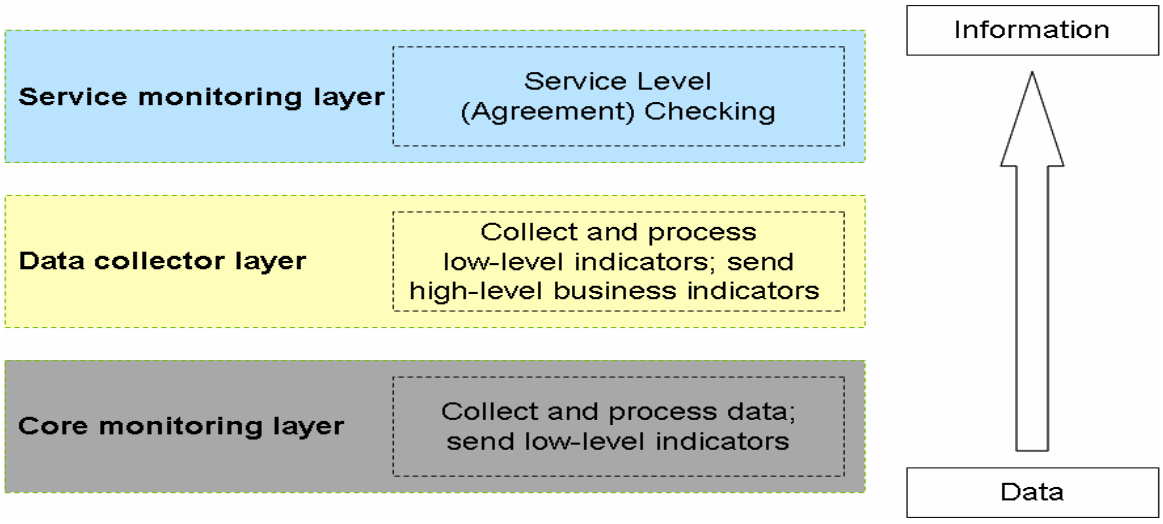


Figure 513: Multi-layered architecture.

6. Security Service

6.1. Security architecture

The security architecture is designed taking into account dependability requirements. This leads us to define the security service as a pluggable external service (**Erreur ! Source du renvoi introuvable.**). It is split into semantic and service layers. The semantic layer is in charge of taking into account security preferences and integrating the community model so that different security strategies can be managed according to the context, while the service layer includes the security mediation (federated identity manager, message security and secure transport.) to support end-to-end technological security services composition and integration in the service chain.

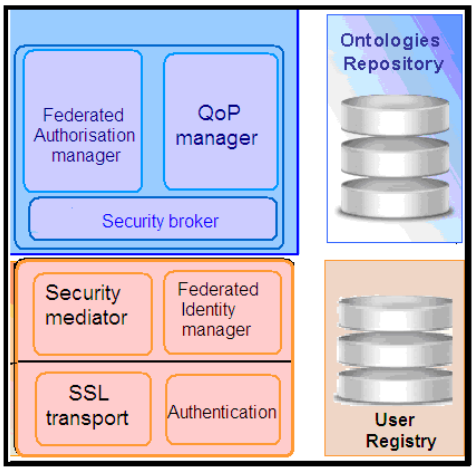


Figure 614: Security Architecture

The dependability requirements lead us to design the security architecture in a distributed way, split among ESB and governance nodes. This distributed architecture is based on a federation manager used to support collaborative authentication and authorisation. Trusted nodes communities are organised in federations so that distributed decisions can be supported.

The semantic level includes three components:

- **QoP manager**: this component is in charge of orchestrating the security components: first it has to activate the QoP matcher, then it has to extract the different actors roles and control that they belong to the convenient access list (generic authorisation process). Lastly, this security agent will activate the service level security mediator in charge of composing the technical security component
- **Federated authorisation manager:** this component has to extract the convenient authorisation strategy from the QoP contract before launching the authorisation process on the different sites involved in the service chain.
- **Security broker:**: This component is used to integrate security requirements while composing a service chain. It supports security integration using a distributed mediation system in which the security policy specification of different services provider may differ, i.e. enforcing security over semantically heterogeneous service provider taking into consideration the requestor requirements and preferences concerning the security. This security broker will manage the security parameters so that technological security component will be instantiated conveniently at the service layer. As said previously, security preferences are organised in different topics:
    - Transport security: this part allows the integration of secured communication protocols to communicate safely on unsafe infrastructure
    - Credential management: this part deals with the authentication and authorisation processes. Access control policy can be either identity-based or role-based.
    - Data security: this part is related to the provider information system infrastructure.

6.2. Service level security

The service layer is split according to 2 sub-levels (**Erreur ! Source du renvoi introuvable.**):

- **Platform-Independent Component** implements the global view on authentication (implemented thanks to a federated identity manager (Single Sign On component) and transport management (used to describe process and deals with the trust chain constraint (i.e. sending and requesting user preferences).
- **Platform-Dependent Component** integrates platform-based certification information, to bind the interface with the convenient communication protocol. It also manages the implemented authentication frameworks and sends the convenient token / login information to the sign-on system embedded in the bus layer.

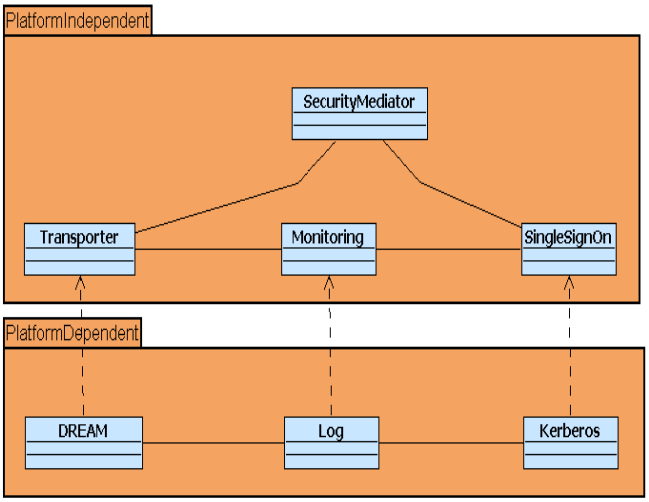


Figure 615: Organisation of the service level

6.2.1. Transport security

The aim of the SSL integration is offering a end-to-end SSL solution between clients and providers who are not necessarily on the same or on a PEtALS node. To achieve this goal we have decided to integrated SSL encryption at two levels of PEtALS,

- between PEtALS nodes: to achieve this goal we decide to add components to DREAM library.
- between the outside and PEtALS nodes: to achieve this goal we decide to modify the SOAP binding component.

with these levels we can achieve our main goal which is offering tools to support a end to end SSL encryption solution. Indeed communications between clients and providers who are on the same PEtALS node do not need to be secured because they are on a same computer and they don't need to use external network to communicate. Communications between clients and providers who are not on a same PEtALS node are now secured thanks to the integration of SSL encryption between PEtALS nodes. Communications between clients and providers who are necessarily on a PEtALS node are secured due to the integration of the SSL encryption between the outside and a PEtALS node.
As Petals transport component is based on DREAM, we integrate the SSL encryption between PEtALS nodes by modifying both the DREAM library and PETALS.
As far as DREAM is concerned, we add five components to the library,

- **SSLProtocolImpl** which represents an SSL encrypted communication.
- **SSLProtocol** which encompasses the SSLProtocolImpl component.
- **ChannelOutSSLStack** which represents an output SSL communication.
- **ChannelInOutSSLStack** which represents an input/output SSL communication.
- **ProtocolChunkRouter** which represents a router based on **ProtocolChunk** chunks. This router selects the output corresponding to the protocol name in the **ProtocolChunk** chunk.

All these modifications have been made as modules so the DREAM architecture hasn't been modified and compatibility with older version is guaranteed.
Then, in PEtALS we modify the following components,

- **DreamWrapper** component, this component now looks for SSL property on exchanges to determine if they must be sent using SSL (adding a ProtocolChunk with SSL tag and selecting the correct port) or not (adding an ProtocolChunk with TCPIP tag and selecting the correct port).

- **DreamTransportProtocol** component, we added an **ProtocolChunkRouter** to determine the protocol (SSL or not) to use for the communication. This component selects the appropriate component for output communication.

Lastly, we have modified PEtALS to be allowed to add the DREAM SSL port in **topology.xml** file. We add a new Router module (**SecurityModule**) to PEtALS this modules will be the binding point between PEtALS and all security mechanisms we will develop.
All these modifications have a small influence in PEtALS architecture and will not alter compatibility with older versions.

Encryption parameters are set up thanks to the addition of keystore elements in the topology.xml file (**Erreur ! Source du renvoi introuvable.**).

```
...
<tns:sub-domain name="subdomain1">
        ...
    <tns:container name="0">
        <tns:dreamservice>
            ...
            <tns:SSL>
                <tns:port>The SSL port</tns:port>
                <tns:keystore>Path to the keystore file</tns:keystore>
                <tns:keystore-password>Keystore password</tns:keystore-password>
                <tns:key-password>The key password</tns:key-password>
                <tns:truststore>Path to the truststore file</tns:truststore>
                <tns:truststore-password>Truststore password</tns:truststore-password>
            </tns:SSL>
        </tns:dreamservice>
    </tns:container>
</tns:sub-domain>
...
```

Figure 616: Topology.XML file description

Note that in general keystore file and truststore file are in the same file, so in truststore properties you have to set same properties as keystore properties.

In order to support a end-to-end secured transport, binding components have also to use a secured transport stack (namely a SSL stack). To set up SSL encryption in SOAP binding component we just have to add the following lines in jbi.xml file (**Erreur ! Source du renvoi introuvable.**).

```
<jbi:jbi version="1.0" xmlns:jbi="http://java.sun.com/xml/ns/jbi"
        xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
        xmlns:soap="http://petals.ow2.org/components/soap/version-3.1">
            <jbi:component type="binding-component"
                bootstrap-class-loader-delegation="parent-first">
            ...
            <soap:ssl-port>The SSL port</soap:ssl-port>
            <soap:keystore>Path to the the keystore file</soap:keystore>
            <soap:keystore-password>Keystore password</soap:keystore-password>
            <soap:key-password>The key password</soap:key-password>
            <soap:truststore>Path to the truststore file</soap:truststore>
            <soap:truststore-password>Truststore password</soap:truststore-password>
        </jbi:component>
    </jbi:jbi>
```

Figure 617: Integration of a secured transport in the binding component

Note that in general keystore file and truststore file are in the same file, so in truststore properties you have to set same properties as keystore properties.

6.2.2. Message security

Message security requires different security services:

- Confidentiality: this service is achieved thanks to a secured transport
- Message integrity: this involves that the message content must be authenticated by the sender and checked by the target service
- Source authentication: the target service must be sure that the message has been really sent by the source service
- Non repudiation: once sent, the source should not repudiate the exchange.

In order to address these requirements, we add a message signature mechanism. This signature mechanism is implemented in the following way: depending on the signature tag, the signing service sign or not the exchange with the source container's private key and put the signature as a part of the exchange properties. In order to allow the destination to check if the message signature, the source Id is also added so that the target service can retrieve the source public key in the trusted key store file ina similar way as for the SSL encryption parameters (see the addition of keystore elements in the topology.xml file (**Erreur ! Source du renvoi introuvable.**)).

This signature is associated to a hashed part of the message and is logged in both source and target log files so that this mechanism implements source authentication, message integrity and message non repudiation services.

6.3. Semantic security: Authentication and authorisation service

6.3.1. State of the art

As seen previously, authentication and authorisations take a geat part in security policy implemtentation. This leads to organise federated identity management. Federated Identity Management aims to share identity information between disparate domains in order to enable seamless cross-organisation authentication and authorisation. In this section we illustrate the representative Federated Identity Management techniques and initiatives in order to highlight their limitations..

Federated Identity Management

The Federated Identity Management is initiated by OASIS and the Liberty Alliance [CCL+08]. The Federated Identity Management enables authenticated identity across multiple organisations. It avoids centralized storage of user information, while allowing users to link multiple local identities related to different accounts. A key element of Federated identity is Circle of Trust that identifies that a user belongs to a community to access certain services.
It broadly covers three specifications [KUMA04]:

1. Identity Federation Framework (ID-FF): allows features for SSO, account linkages, anonymity, affiliations and options for meta-data exchange.
2. Identity Web Services Framework (ID-WSF): allows features for Permission Based Attribute Sharing, Identity Service Discovery, Interaction Service Security Profiles and Identity Services Templates.
3. Identity Services Interfaces Specifications (ID-SIS): allows for interoperable services to be built over ID-WSF. Interoperability is offered through use of context dependent agreed schemas.

As we observe the trust notion is fundamental to deal with identity federation. To this end, the SOA Reference architecture proposed by the OASIS [OASI08] proposes a trust model between services stakeholders and service customersIn the next following paragraph we present an abstract description of OASIS trust model.

OASIS Reference Architecture Trust Model

The OASIS Reference architecture proposes a trust model between services stakeholders and service customers in order to define who can perform an action or raise an event. The Trust Domain is defined to model abstract concepts over policy-based trusted social groups. It is defined as *"An* abstract space of actions which all share a common trust requirement; i.e.,

all participants that perform any of the actions must *be in the same trust relationship"*. This leads to define several trust level associated to different "social groups" mixing stakeholders and participants and trust relationships. Trust relationships between participants can be used to derive decentralised authentication and authorisation so that trust chain can be established for a global service chain.

Such a credential-based security mechanism expresses security choices that are then used to define security mechanism to enforce security. To be able to communicate these choices, they can be described in policies. Policies are defined as a set of rules that can be tested by decision points whereas enforcement points are used as relays in the distributed system.
Many projects and initiatives have created architectures and open-source technologies and/or specifications for federated identity-based authentication and authorization like Shibboleth [CEH+05] OpenAthens[1], OpenID[2] , CAS [CAS04]; among these works we can distinguish Shibboleth which has the advantage on other architectures by introducing the notion of Where Are You From (WAYF) service. This component allows providing seamless access control for collaborating organisations by supporting a flexible cross-organisation authentication mechanism, while allowing enforcing local authorisation policies to control what resources authenticated users are allowed access to.

Shibboleth

Shibboleth is an Internet2 middleware supports the establishment of federated authentication and convey security attributes across independent organisations [CEH+05]. User authentication at a local Identity Provider (IdP) can support Single Sign-On across a federated organisations where security attributes and assertions are released and used by service providers (SP).

The key element of Shibboleth is trust; through Shibboleth, each service provider has pre-established trust relationships with a set of Identity providers (IdPs). Thus, end users having single usernames and passwords from their home organisation are provided with seamless access to various service providers resources. The Shibboleth architecture consists of Identity Providers, Service Providers and Where Are You From (WAYF) services. When a user attempts to access a service he/she is redirected to a WAYF server that asks the user to pick his home (IdP) from a list of known and trusted service providers. The user then is redirected to his/her authentication server. The home site redirects the user back to the SP while providing a digitally signed SAML [SAML05] (Security Assertion Markup Language) authentication assertion message i.e. token, asserting that the user has been successfully authenticated and providing the SP with a temporary pseudonym for the user (the handle), the location of the attribute authority at the IdP site and the resource URL that the user was previously trying to access. The digital signature on the SAML authentication assertion is verified, the resource site then returns the handle to the IdP's attribute authority in a SAML attribute query message, depending on which the IdP's attribute authority returns a signed SAML attribute assertion message (called Attribute Certificate AC). Depending on the attributes assertions, SPs within a federation are still autonomous and are able to decide whether the provided attributes are sufficient for access resources and which attributes they are able to release to an SP according to the degree of trust it keep with them.

As explained above achieving authentication and authorisation according to *Shibboleth* needs passing tokens containing user attributes and credentials using SAML, SAML is a standard XML based language and protocol thought to exchange security assertions.

SAML

SAML assertions help in security context attributes transmission, including user attributes. SAML assertions are created to convey the security context to other entities that depend on this context for their security or business logic. The entity that manages security communicates the security context to all nodes in the message path. SAML fulfils these requirements as it provides a language for expressing the service security context. To do so different statements are defined:

1. Authentication Statement: asserts authentication results, after a service authenticates a user.
2. Attribute Statement: asserts user attributes, for instance validating the username and password or asserting the identity of the user. It can assert more information, such as the groups the user belongs to, the user's preferences, and the user's location..
3. Authorization Decision Statement: asserts the authorization decisions by conveying the kind of access granted to the user for given resources.

[1] OpenAthens framework, available at www.athensams.net

[2] OpenId framewofrk available at http://openid.net/

In the stat of the art we exposed the basic and most commonly used models and techniques to implement federated identity solutions. In the next section we introduce our approach along with the implementation through PEtALS.

RBAC integration

The definition of a security strategy leads to deploy security policies to control access based on actors' positions within the enterprise organisation structure i.e. their organisational roles, these roles can be integrated into enterprise workflow specifications and user authentication systems [LHB02, LB00]. Role-Based Access Control (RBAC) [FCK95] is a security model which relies on the enterprise organization view to provide authorization to access resources. The RBAC model creates an indirect relationship between rights and actors through the roles played by the actors. Thus, security policies are defined to govern roles, and facilitate the management of security policies. RBAC affects users of the roles they can play; these roles are associated with permissions which provide access to resources [CCW+07]. This model allows establishing high level organizational policies by adding constraints on the top of the relations between its components. It allows also for each principal to get multiple roles, and then choose to activate one or more of the assigned roles during each session. However, the RBAC model does not take in consideration the notion of execution context. To do so, the work in [WRWR05] extends the RBAC for adaptive process management systems by defining new type of access rights –in addition to user dependent access rights– called in this work process-dependent access rights, this allows changing permission according to the current executed process. The DRBAC [ZP06] proposes dynamic roles to access resources by defining *policy context* sets which allow triggering changes in role state via *role state machines* according to context changes. The Generalized Role-Based Access Control (GRBAC) model [CMM01] extends the role notion to be applied to all system entities i.e. s*ubject roles, environment roles,* and *object roles.* Contextual information is checked to grant access according to entities roles. However, both DRBAC and GRBAC do not consider the collaborative business context where dynamic ad-hoc groups of users of different policies and contexts conjointly work to carry out a common objective. Additionally, due to the multitude of system entities, management of roles in GRBAC becomes difficult to maintain. The work in [RKL06] extends the RBAC model with the physical location of resources and actors to decide whether a role has an access right to resources. [YLS96] Proposes role-based access control security architecture to support distributed and multi-organisational enterprise. Though this work presents well-designed and integrated policy-based security architecture, it suffers from some drawbacks: the notion of context has been neglected which, in this case, impacts the totality of the architecture. Additionally, the work treats the problem from a purely security-oriented viewpoint without taking into account the business dimension such as process or workflow and their related problem of coordination and dynamic configuration. In this order, the RBAC model is extended to the W-RBAC [WKB07] by adding the "doer" relationship as well as various dynamics to monitor the integrity of workflow. Lastly, Roles can also be added in the BPEL task code so that authorisation can be verified [XCK06]. Despite of the specification of several roles in the BPEL4RBAC architecture[WZSY08], this extended role notation does not take into account role semantic relationships. To overcome this limit, semantic annotations must be added.

Discussion

SSO enables creating composite services and securely sharing distributed resources by adopting a cross-organisation authentication mechanism as single set of user security credentials are sufficient to allow access to a multitude of federated resources across the ESB. It allows in the same time local enforcement of access control policies. However, the challenge is how to establish a composite service in a dynamic manner where sets of fine grained distributed security authorization policies of independent organisations and which do not keep a direct trust relations in each other can be supported. In SOA context composite services can be dynamically established linking disparate services at run time when no prior knowledge about new added services and users etc.

Shibboleth is more static than this vision, instead Shibboleth requires having a static list of well known and trusted IdPs to manage the authentication decision at a given SP. The problem is that Shibboleth model requires pre-known partners (static list of trusted IdPs) to be defined. This goes against the "on-the-fly" composition of services and limits the deepness of the chain of service as a provider can not authenticate users unless they belong to those providers with which it keeps a direct trust relation and where new enterprises or users are brought together for short time collaboration. This is not realistic for evolving SOA. It should be possible to delegate the privilege to trusted SP to issue locally recognized attribute certificates for their trusted partners. This is especially useful when complex or short lived dynamic Services are to be established and managed.

To address this, we propose to slightly modify Shibboleth model by keeping on each SP (in addition of traditional trusted IdPs) a list of delegated issuing IdPs (DIdP). The DIdP is an IdP that can issue Attribute Certificates for trusted users belonging to other domains than the DIdP itself belongs. Thus DIdP itself can issue ACs to their subordinates or partners. To face the security risks arising from such scenarios the SP will restrict the privileges that users -trusted by its DIdPs- can hold. The DIdP also can delegate to other entities it trusts, to also delegate privileges to others. Again, to minimize the potential security risks that might arise through this, subordinate DIdP will always have lower privilege than their superiors. Through this, Sps are able

to establish a composite service dynamically by creating attribute certificates for their subordinates SPs associated with the particular demands. Once defined, users from subordinates SPs wishing to access resources across multiple organisation are able to use the single sign-on capabilities of Shibboleth to authenticate themselves at their home IdP, and have these attributes (which have been dynamically created) to be used by SPs to make subsequent authorization decisions.

N.B: This can be achieved by roles; ACs can consist of digitally signed certificate including four fields <Role, Target, Action, attributes>.

Thus when in this new paradigm, a user attempts to access a Shibboleth protected service he is redirected (as usual) to a WAYF server that asks the user to pick his home Identity Provider (IdP) from a list of known and trusted service providers, if the user does not have a direct trust relation with the SP he will be asked to provide his home IdP as well as to select from the DIdPs list the one with which he has a trust relation.

After the user has provided his home IdP and has picked the selected DIdP, his is redirected to the selected DIdP server while providing the DIdP with the requestor's IdP. Starting from this point the ordinary Shibboleth mechanism is followed.

If the user does not select any DIdP from the list, the WAYF server diffuses to all DIdPs in the list the URL provided by the requestor to inquire if one of them or of their subordinates has a trust relation with the mentioned IdP, if there is a positive answer the requestor will be redirected to That DIdP and the normal process restart again (at the WAYF server associated to the DIdP).

6.4. Distributed Security Service Organisation

The distributed security service manager is in charge of the authentication and credential management (Figure 31**Erreur ! Source du renvoi introuvable.**). This system is organised in a distributed way: each node belonging to the trusted community (a node can be associated to a PETALS or a DRAGON instance) represents a realm and is associated to an instance of the security service. Each instance includes the following components:

- The federation manager manages the nodes community. It has to manage the community topology, send and process interactions with the other realms belonging to the same community.
- The authentication service implements the Single Sign On system. It includes and Identity provider and authentication service as well as a token manager in charge of generating the convenient tokens after the user has sign-in.
- The attribute manager is in charge of getting the convenient information from a user registry.
- The credential management service has to manage the interface with the service repository (implemented thanks to DRAGON in our case). It has to compose and orchestrate the authorisation process including the service authorisation policy acquisition, the service customer authentication and its convenient attributes collection before setting the matching process.

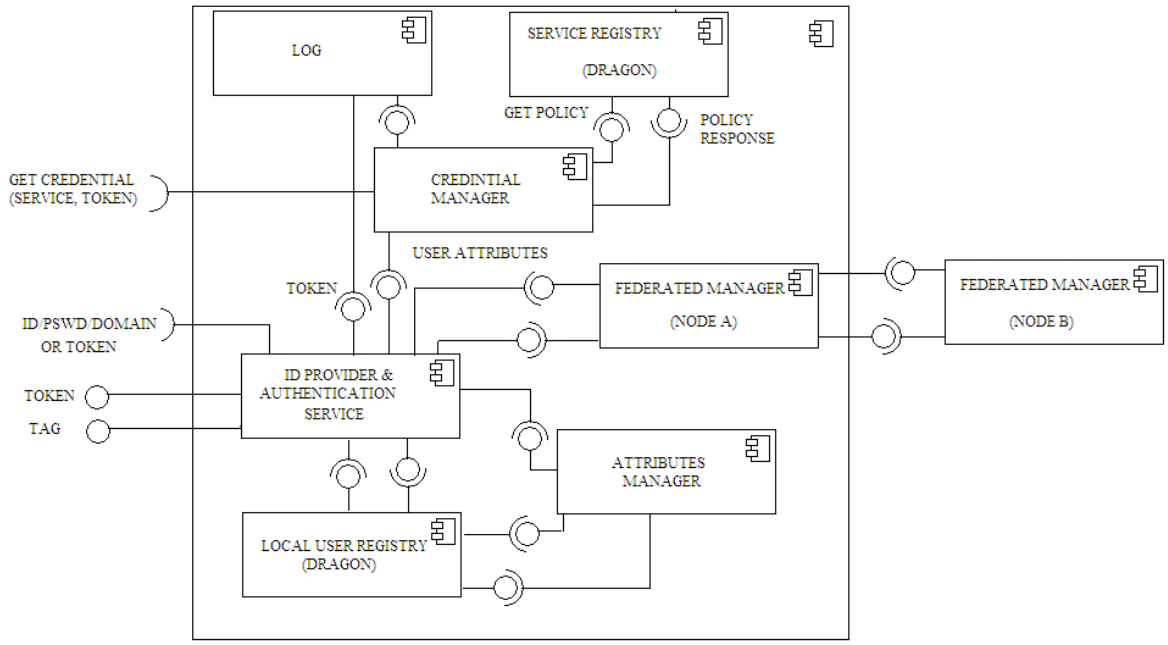


Figure 618: Security component organisation

6.4.1. User registry

In our architecture, the user is associated to a service storing its private information. As proposed in the OASIS reference architecture, a user is associated to a main identity and can use several identifiers (seen as secondary identity information) depending on the trusted community it is involved in (**Erreur ! Source du renvoi introuvable.**). The user also owns different attributes (roles…) that can be used to set access authorisation to a service or to a given piece of information (credential management).

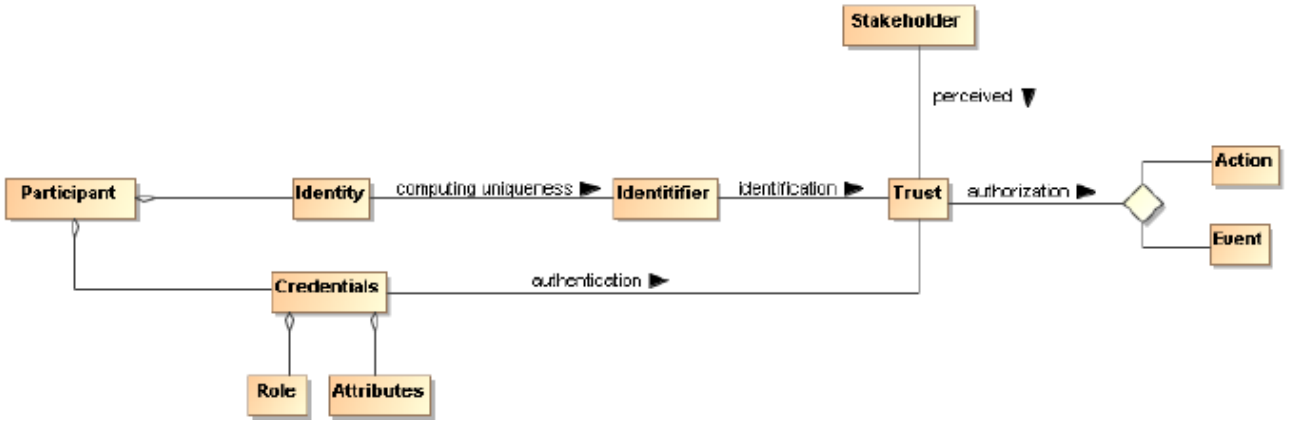


Figure 619: Authorisation model taken from [OASI08] p. 90

Our architecture implements partly this model. Each user is associated to a main identity (defined as a "subject") associated to the authentication data that will be used to sign in depending on the authentication library (login / password in our prototype). Then secondary identities (called "Principals") are used to organise the different identifiers and the related attributes so that different roles (depending on the reference model) can be used to annotate semantically the user registry (**Erreur ! Source du renvoi introuvable.**). By this way, authorisation models can be added in the service discovery process as user related functional properties selection.

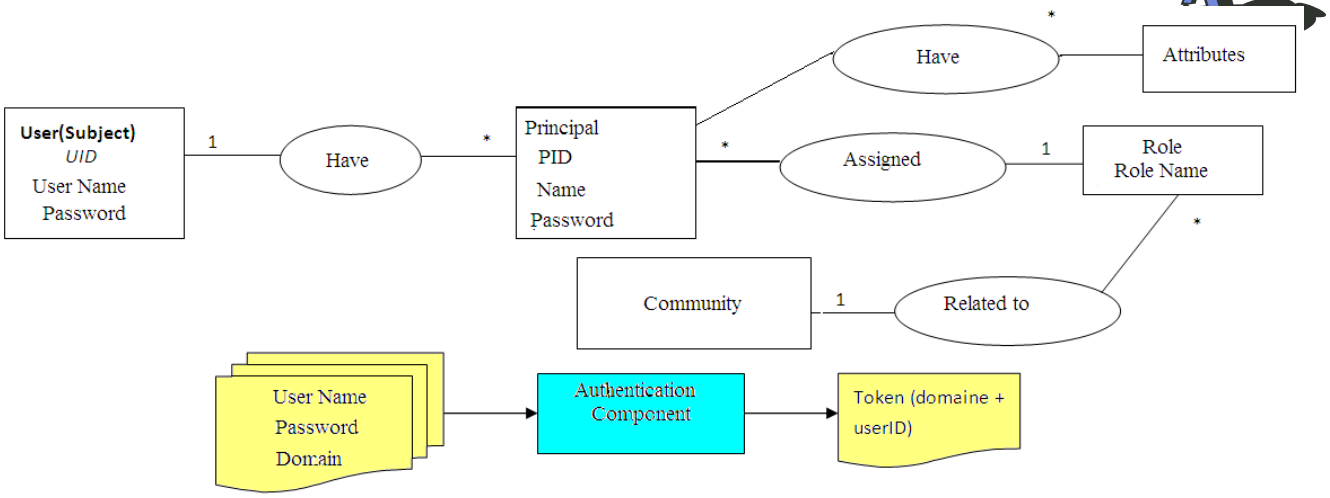


Figure 620: Internal user registry organisation

6.4.2. Distributed authentication system

Different authentication strategies can be set (**Erreur ! Source du renvoi introuvable.**). In our architecture different strategies can be used from the local authentication to a distributed single sign on system. The SingleSignOn architecture is organised in realms. One realm is attached to each PEtALS node. In each realm, the SSO repository includes both realm authentication information (including trusted 3rd party references) and user information. As a person can be associated to several roles, different authentication information can be stored. Connection between realms is set thanks to the federation manager component. This component is used to check if the user is already registered or not in a referenced realm. Trust relation between realms is used to identify potential and trusted partners to exchange actors' information with. Once registered in the SSO directory, a token is sent back). By this way, identity information can be stored and resent according to the needs. Of course, users can also "disconnect" and delete personal information thanks to a "signoff" process.

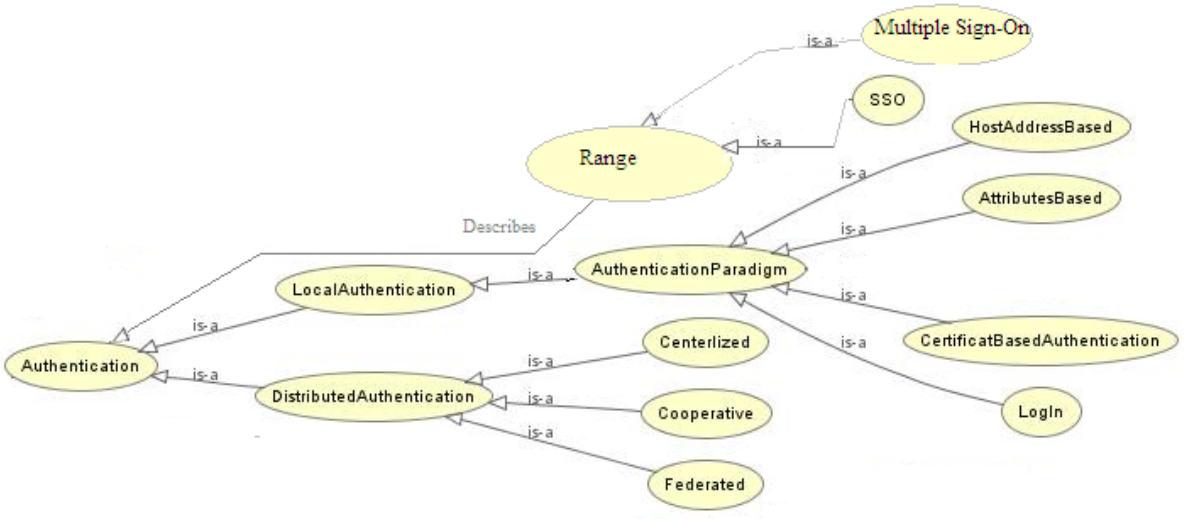


Figure 621: Authentication strategy ontology

The authentication process ("sign-in" function) is achieved once the user opens a PETALS or a Dragon Session. As in Shibboleth, this connection process includes a "WAYF" (Where Are You From) part. Consequently, the user has to provide its main identity information including authentication proof (the password in our case) AND the realm identification (namely the node storing its personal information). These information are sent locally on the IdProvider. Depending on the domain the user belongs to, the IdProvider either processes the connection (**Erreur ! Source du renvoi introuvable.**) or sends it to its federation manager which is in charge of routing it on the convenient node IdProvider via its local federation manager. (**Erreur ! Source du renvoi introuvable.**)

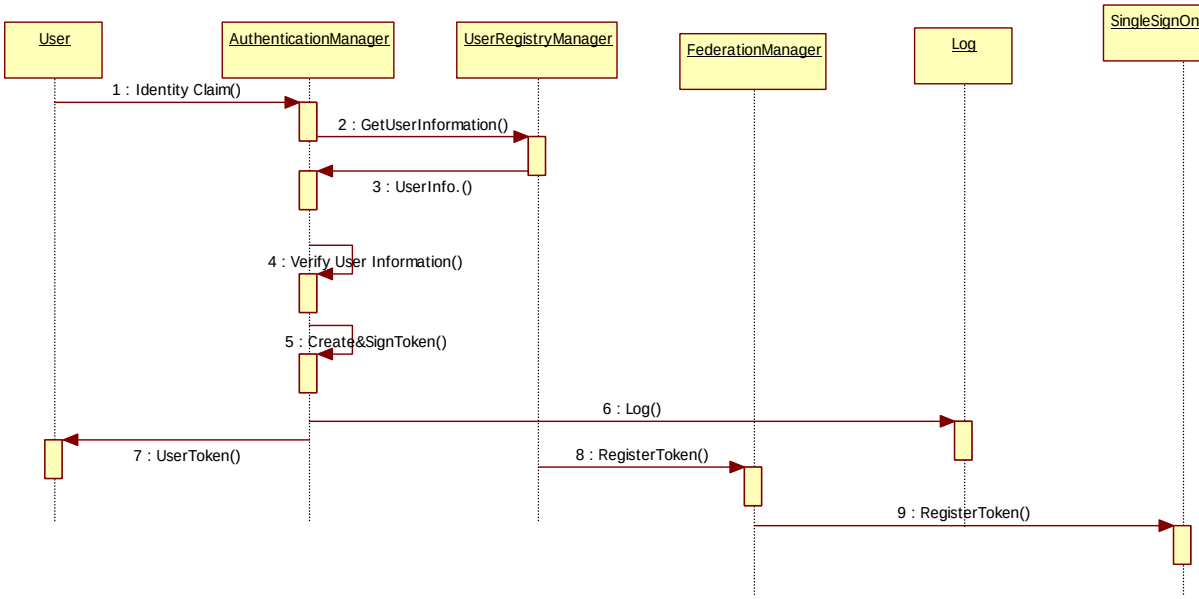


Figure 622: Local authentication process

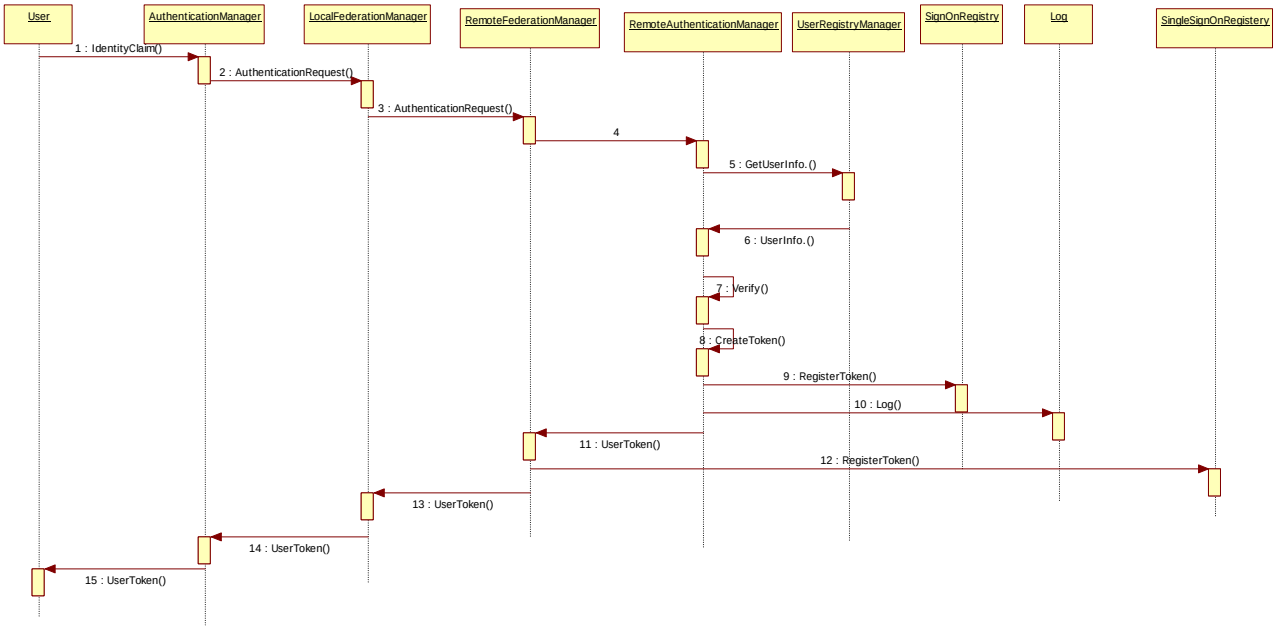
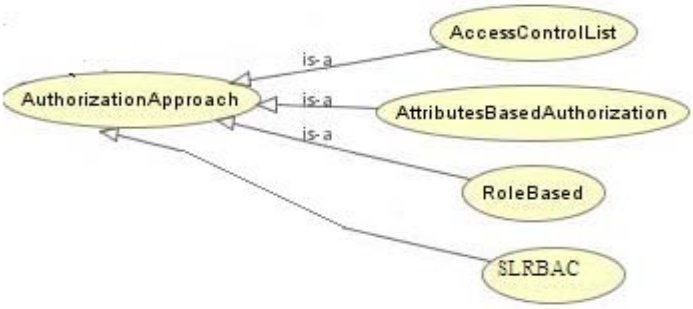


Figure 623: Distributed authentication process

Once logged in, a token is sent back to the requestor. This token consists in the user identity information (main id and domain id) and a signature achieved by the IdProvider which is a connection proof. The connection information is also logged in (token and connection time stamp) in the log system so that it can be retrieved if necessary.

A Sign Out mechanism is also provided to allow the user to disconnect from the system. By this way, the connection token is removed from the active tokens registry. The disconnection event is also logged in (**Erreur ! Source du renvoi introuvable.**).



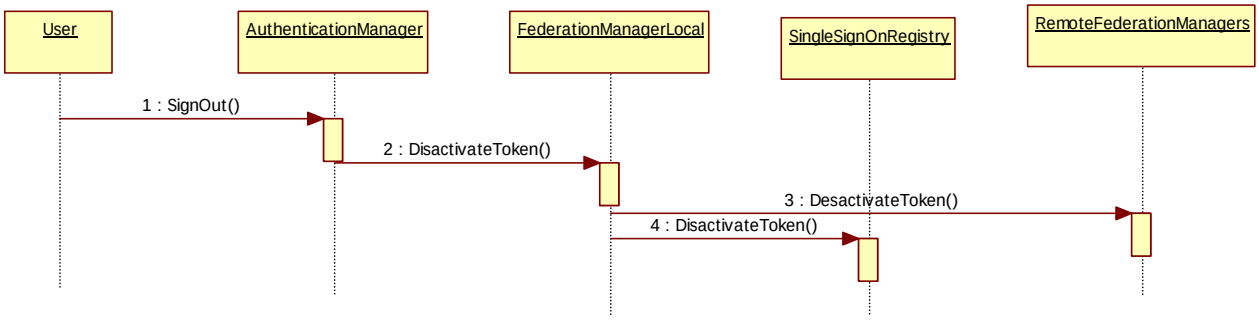


Figure 624: Sign Out process

6.4.3. Credential management

Different authorisation policies can be set to allow or not the service invocation. These policies can refer either to an access control list (based on user identifiers), a particular attribute or a role based access control (**Erreur ! Source du renvoi introuvable.**). After the service has been discovered and before the service is invoked, the credential manager is used to log the authorisation. It has first to extract the convenient authorisation policy from the service registry. Depending on the authorisation strategy, the credential manager has to get the convenient attribute from the attribute manager. Thanks to the login token transmitted by the requestor, the credential manager invoke the attribute manager (directly for a locally authenticated user or via the federation manager service) to get the requested attribute before the matching process. If the user attribute matches the access control criteria, a signed time-stamped access token (including the user identity, a time stamp and the requested attribute) is logged in and sent back to the requestor (**Erreur ! Source du renvoi introuvable.**). The attribute collection process is also published so that services can invoke directly this service in order to get the requested attributes they need for their own internal authorisation processes (**Erreur ! Source du renvoi introuvable.**)

Figure 625: Authorisation ontology

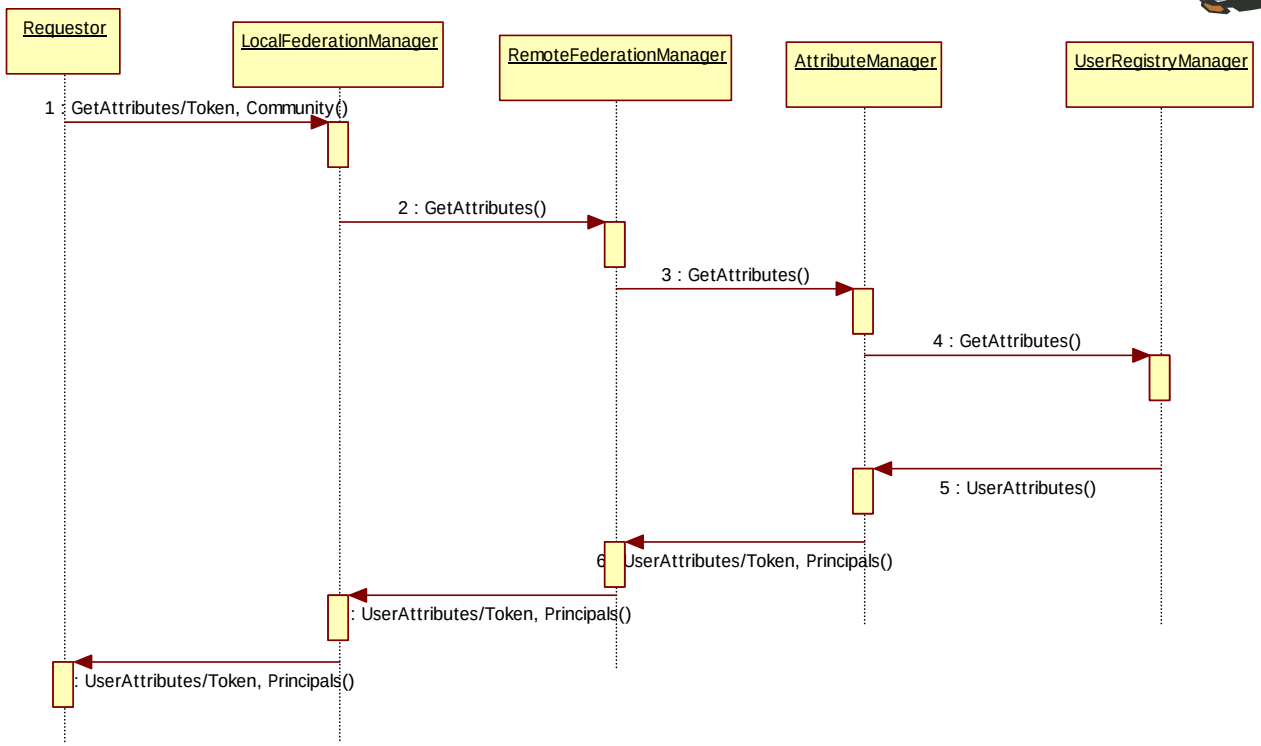


Figure 626: Distributed authorisation process

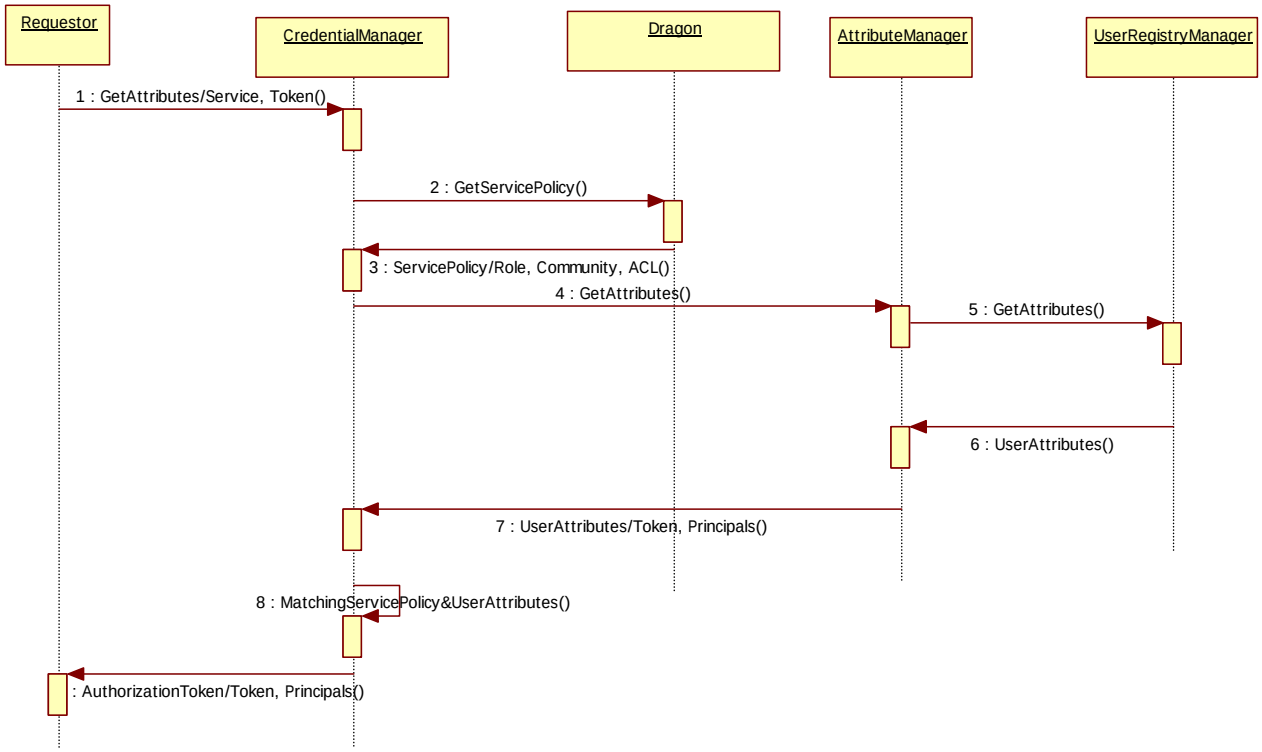


Figure 627: Attributes collection process

6.5. Integration of the Security service bus

The service security bus has been designed as an external component. As far as the semantic layer is concerned, this component is implemented as an independent service proposing different interfaces (**Erreur !**

Source du renvoi introuvable.). As far as the service layer is concerned, the security service has to be included in the node instance. As presented section 5.2, security patterns are used to set security tags (configured in a XML file used as a parameter for the service invocation) (**Erreur ! Source du renvoi introuvable.**) so that a convenient routing to secured / non secured mechanisms can be addressed in the bus (**Erreur ! Source du renvoi introuvable.**). Presents globally

The presents globally the interaction between PETALS and Dragon nodes with the bsecurity service at runtime.

Interface	Short description	Operation(s)
SecurityHandler	Interface for selecting the convenient security services.	**setServiceSecuredTransport(service)**: Tag the secured transport metadata field in the service interface
AuthenticationManager	Requests and verifies user information from UserRegistryService/or FederationManager.	**SignIn(User, Token):** authenticate the user and import its personal data in the internal SSO directory
SignOutModule	Manages SSO registry, add or erase tokens	**SignOut(User, Token):** Export authenticated user personal data to the user registry and remove it from the SSO repository
FederationManager	Communicate user token obtained from other domains' federation manager.	**getUserIdentity(Token, authenticationtype):** retrieve the convenient user authentication information for the registered user identified by the token in the SSO repository.
UserRegistryService	Communicates user information to requesting services.	**GetUserInformation, GetUserAttributes:** retrieves user information and user attributes and send them to requesting services.
CredintialManager	Manages authorization process by matching service policy and user attributes	**GetUserAuthorisation(Token, service):** retrieve the service authorization parameters and control the matching with the properties associated to the registered user (identified by the token)

Figure 628: Security service interface specification

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/SecurityPreferences"
    xmlns:tns="http://xml.netbeans.org/schema/SecurityPreferences"
    elementFormDefault="qualified">
- <xsd:simpleType name="wssecurityType">
  <xsd:restriction base="xsd:anyURI" />
    </xsd:simpleType>
- <xsd:simpleType name="authenticationType">
  <xsd:restriction base="xsd:boolean" />
    </xsd:simpleType>
- <xsd:simpleType name="authorizationType">
  <xsd:restriction base="xsd:boolean" />
```

```
    </xsd:simpleType>
- <xsd:simpleType name="encryptionType">
  <xsd:restriction base="xsd:boolean" />
    </xsd:simpleType>
- <xsd:simpleType name="secureTransportType">
  <xsd:restriction base="xsd:boolean" />
    </xsd:simpleType>
- <xsd:simpleType name="digitalsignatureType">
  <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
- <xsd:simpleType name="tokenType">
  <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
- <xsd:complexType name="AccountType">
- <xsd:sequence>
  <xsd:element name="token" type="tns:tokenType" />
  <xsd:element name="subject" type="tns:subjectType" />
  <xsd:element name="principal" type="tns:PrincipalType" />
  <xsd:element name="IsPrincipal" type="tns:IsPrincipalType" />
    </xsd:sequence>
    </xsd:complexType>
- <xsd:simpleType name="uIdType">
  <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
- <xsd:complexType name="subjectType">
- <xsd:sequence>
  <xsd:element name="uid" type="tns:uIdType" />
  <xsd:element name="domain" type="tns:domainType" />
    </xsd:sequence>
    </xsd:complexType>
- <xsd:complexType name="PrincipalType">
- <xsd:sequence>
  <xsd:element name="uid" type="tns:uIdType" />
  <xsd:element name="domain" type="tns:domainType" />
    </xsd:sequence>
    </xsd:complexType>
- <xsd:simpleType name="domainType">
  <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
- <xsd:simpleType name="IsPrincipalType">
  <xsd:restriction base="xsd:boolean" />
    </xsd:simpleType>
- <xsd:complexType name="SecurityPreferencesType">
- <xsd:sequence>
  <xsd:element name="account" type="tns:AccountType" />
- <xsd:element name="requirements">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element name="authentication" type="tns:authenticationType" />
  <xsd:element name="authorization" type="tns:authorizationType" />
  <xsd:element name="secureTransport" type="tns:secureTransportType" />
  <xsd:element name="encryption" type="tns:encryptionType" />
  <xsd:element name="signature" type="tns:digitalsignatureType" />
    </xsd:sequence>
```

```
  </xsd:complexType>
  </xsd:element>
<xsd:element name="wssecurity" type="tns:wssecurityType" />
  </xsd:sequence>
  </xsd:complexType>
<xsd:element name="securityPreferences" type="tns:SecurityPreferencesType" />
  </xsd:schema>
```

Figure 629: XML configuration file

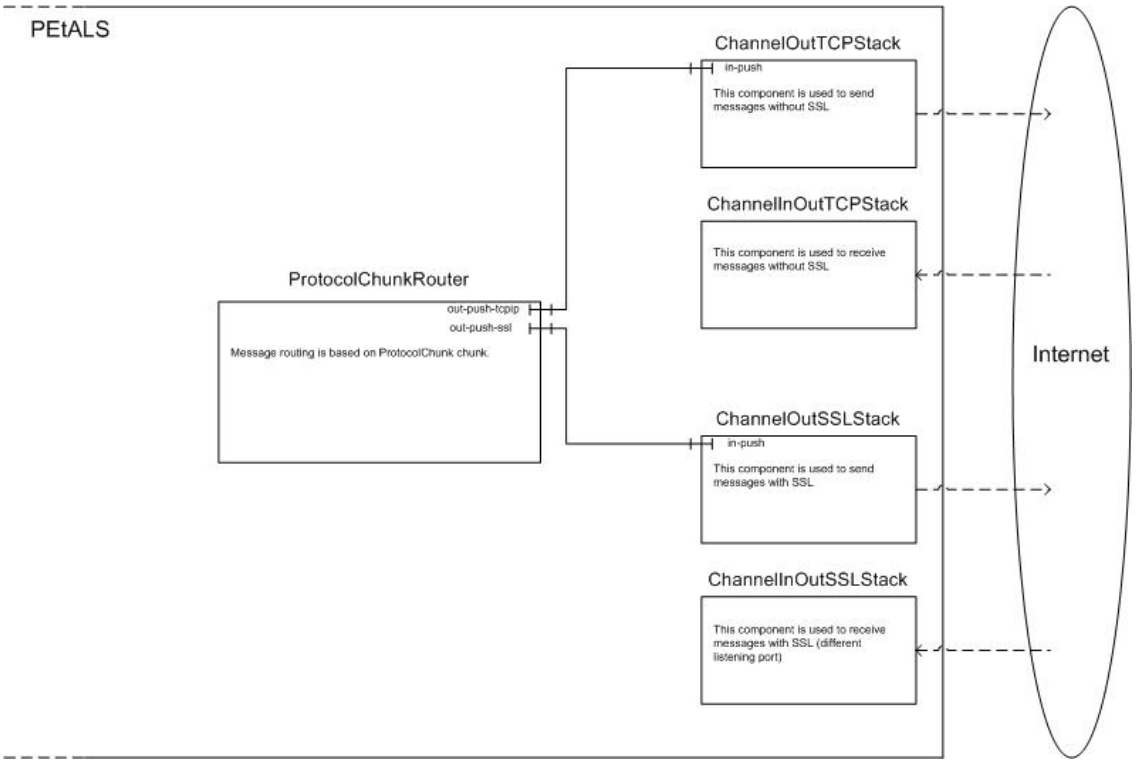


Figure 630: Integration of the secured transport

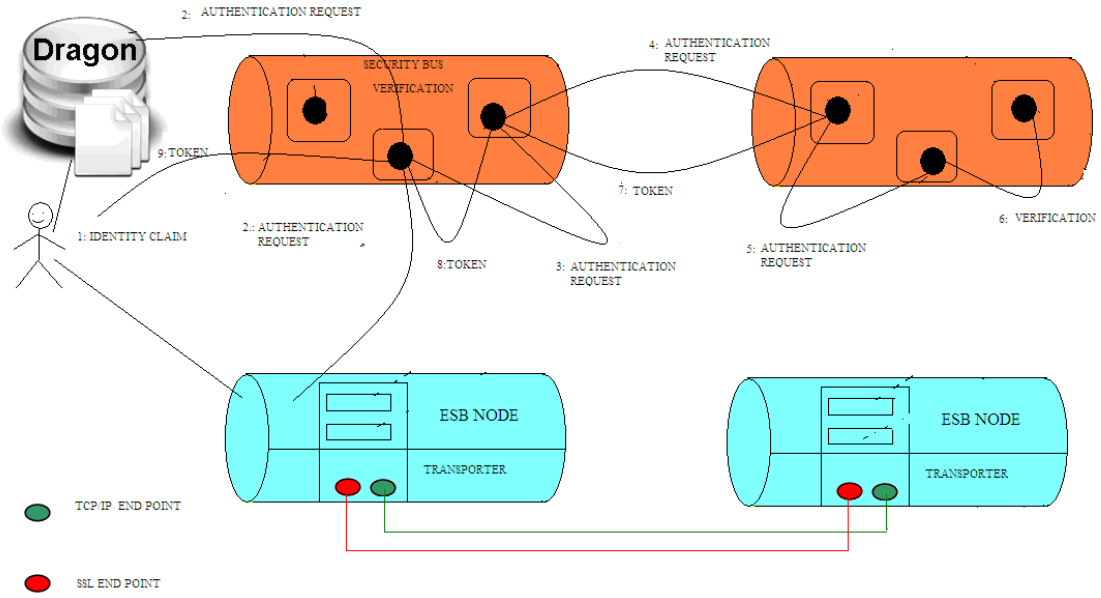


Figure 631: Integration of the security service

7. Conclusion

In conclusion, this deliverable explains how use non-functional properties at runtime. The non-functional properties are used to verify the global QoS of BPEL process or detect violation in service level objectives at run-time.

8. References

[WSDM, 2006] OASIS Web Services Distributed Management (WSDM) TC, www.oasis-open.org/committees/wsdm/, 2006.

[WS-RM, 2007] OASIS WS-ReliableMessaging Policy Assertions 1.1, http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.html, 2007

[Ould Ahmed M'Bareck & Tata, 2008] Ould Ahmed M'Bareck, Nomane, & Tata, Samir. 2008. Services Web : revue des approches de description sémantique. Conférence internationale "Systèmes d'Information et Intelligence Economique", Hammamet, Tunisie.

[W3C, 2001] W3C. 2001. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl/.

[WS-PolicyAttachment, 2006] W3C Member Submission "Web Services Policy 1.2 - Attachment", April 2006 http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/

[WS-SecurityPolicy, 2007] OASIS WS-SecurityPolicy 1.2, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.html, 2007

[ACD+07] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification. Open Grid Forum (OGF), Grid Resource Allocation Agreement Protocol (GRAAP) WG, March 2007. http://forge.gridforum.org/sf/projects/graap-wg.

[FK98] S. Frølund and J. Koistinen. QML: A Language for Quality of Service Specification. Technical Report HPL-98-10, Software Technology Laboratory, Hewlett-Packard, February 1998.

[vdV04] Eric van der Vlist. Relax NG. O'Reilly, 2004.

[WS-QM09] OASIS Web Services Quality Model (WSQM) TC: http://www.oasis-open.org/committees/wsqm/charter.php

[CAS 04] Central Authentication Service, Yale University Technology and Planning, http://tp.its.yale.edu/confluence/display/TP/Home

[CCL+08] Cahill C., Canales C., Le Van Gong H.A., Madsen P., Whitehead G., 2008. Liberty alliance web services framework: a technical overview version 1.0. Liberty Alliance white paper.

[CCW+07] Chen T.Y., Chen Y.M., WangC.B., Chu H.C., Yang H., 2007. Secure resource sharing on cross-organization using a novel trust method. Robotics and Computer-Integrated Manufacturing Vol. 23, pp. 421-435.

[CEH+05] Carmody S., Erdos M., Hazelton K., Hoehn W., Morgan R.L., Scavo T., Wasley D., 2005. Shibboleth architecture: protocols and profiles. Shibboleth white paper available via http://shibboleth.internet2.edu

[CMM01] Covington J.M., Moyer J.M., Mustaque A., 2001. Generalized Role-Based Access Control for Securing Future Applications, Proc. (ICDCS'01), pp. 391-401.

[FCK 95] Ferraiolo D., Cugini J., KuhnR., 1995. Role Based Access Control: Features and Motivations, Proc. Annual Computer Security Applications Conference, pp. 554-563.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. Journal of intelligent information systems, 8(2) :117–132, 1997.

[KUMA 04] Kumar A., 2004. Federated Identity Management, TechTarge white paper, 2004 available from: http://www.securitydocs.com/library/2782

[LB 00] Lin A., Brown R., 2000. The application of security policy to role-based access control and the common data security architecture. Computer Communications, Vol.23, pp. 1584-1593.

[LHB02] Li D., Hu S., Bai S., 2002. A uniform model for authorization and access control in enterprise information platform. Proc. EDCIS2002, Lecture Notes in Computer Science, vol. 2480, pp.180-192.

[OASI08] OASIS Service oriented architecture reference model TC, 2008. Reference architecture for service oriented architecture. Version 1.0. 104 Pages. Available online at http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

[Ric09] Elias RICKEN DE MEDEROS, "Atelier dirigé par les modèles pour l'assemblage de médiateurs ", Université Joseph Fourier (Grenoble), Laboratoire LIG, juin 2009.

[RKL 06] Ray I., Kumar M., Lijun, Y., 2006. LRBAC: A Location-Aware Role-Based Access Control, Proc.ICISS2006, Lecture Notes in Computer Science Vol. 4332, pp. 147-161.

[SAML05] OASIS, Security Assertion Markup Language (SAML). Available from:<http://xml.coverpages.org/saml.html>.

[Wie92] Gio Wiederhold, "Mediators in the architecture of future information systems", Dept. of Computer Science, Stanford University, California, travail publié dans IEEE Computer, Vol. 25, Issue 3, pages 38-49, Mars 1992.

[WKB 07] Wainer J., Kumar A, Barthelmess P., 2007. DW-RBAC: A formal security model of delegation and revocation in workflow systems, Information Systems, Vol. 32, pp. 365-384.

[WRWR05] Weber B., Reichert M., Wild W., Rinderle S., 2005. Balancing Flexibility and Security in Adaptive Process Management, proc. CoopIS, DOA, and ODBASE 2005, Lecture Notes in Computer Science, Vol. 3760, pp. 59-76.

[WZSY08] Wang X., Zhang Y., Shi H., Yang J., 2008. BPEL4RBAC: An Authorisation Specification for WS-BPEL. in J. Bailey et al. (Eds.): WISE 2008, LNCS 5175, pp. 381–395,

[XCK06] Xiangpeng Z., Cerone A., Krishnan P., 2006. Verifying BPEL Workflows Under Authorisation constraints. In S. Dustdar, J.L. Fiadeiro, and A. Sheth (Eds.): BPM 2006, LNCS 4102, pp. 439–444,

[YLS 96] Yialelis N., Lupu E., Sloman M., 1996. Role-Based Security for Distributed Object Systems, Proc. ICE'96, pp. 80–85.

[ZP 06] Zhang G., Parashar M., 2006. Dynamic context aware access control for grid application, Proc. of GRID'XY, Computers & Security, Volume 25, pp. 507-521.