034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D1.8

Network Service Plane Enhancements

Due date of deliverable: 2008-09-30
Actual submission date: 2008-09-30
Document code: <Phosphorus-WP1-D1.8v1>

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
Rheinische Friedrich-Wilhelms-Universität Bonn

Revision 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	×
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

Since its first prototype presented in Deliverable D1.5, some major enhancements have been added to the Network Service Plane implementation. These include possibilities of distributing the interdomain functionality to improve failure resilience and scalability, integration of AAA concepts developed within WP4, multi-constraint support, malleable reservations for data transfers, a notification framework to eliminate the need for polling status changes in regular intervals, and some other minor enhancements. This deliverable describes the new prototype.

Table of Contents

Table of Figures

0 Executive Summary

This document describes the advances made in the Network Service Plane implementation concerning a variety of issues. It describes the new features that have been added and gives a description on how they have been implemented.

1 Foreword about the Harmony system

Work package 1 prototypes have demonstrated to be a proof of concept of the feasibility of Phosphorus' challenges in multi-domain lightpath provisioning. These prototypes have achieved such a stability and maturity that has made necessary the adoption of a branding name. Thus, WP1 prototypes of the Network Service Plane, the NRPS Adapters and the service interface have been located under the umbrella of the **Harmony system**.

Moreover, with this new branding name, WP1 prototypes, achievements and demonstrations can be more effectively disseminated. Working teams in WP1 have considered several options for the branding name. However, *Harmony* positioned itself as a clear, simple name easy to remember. Moreover, the name *Harmony* transmits the idea of orchestration and harmonization, which is precisely what Phosphorus project aims to settle among the heterogeneous Network Resource Provisioning Systems all around the world by means of WP1 prototypes.

Apart from the branding name, WP1 prototypes have received new acronyms for its internal entities which easily allow identifying layers in Harmony's architecture. These new acronyms do not differ from the original proposal in D1.1, but are more specific, as will be detailed in the following lines.

New Harmony acronyms

Originally, D1.1 defined only one acronym: Network Service Plane. Nevertheless, it became too generic and vague when the complexity of the architecture arose. Moreover, this name referred to a software entity in WP1's first prototype, which could be misleading. At the current stage, NSP is an acronym for a real plane populated with one or more entities called Inter-Domain Broker (IDB).

Other independent software entities existed on stage, such as the NRPS Adapters, but no acronym was set for them. In order to provide them with an identity mark, their name was changed to the acronym HNA (Harmony NRPS Adapter). This way, the branding name appears and the original (descriptive) name is kept. As HNAs are set on top of each NRPS system to interoperate with the NSP, they depict a thin layer, which also received name and acronym: the Harmony Adaptation Layer or HAL.

Finally, one of the most important developments within Phosphorus WP1 needed also a name (and acronym): the service interface. Again, the branding name was considered and the service interface was named the Harmony Service Interface or HSI.

Figure 1.1 shows a simple Harmony set up with one IDB at the NSP and one NRPS which is being connected to the NSP by means of one HNA (only one HNA allowed per NRPS). Names in the arrows are detailed hereby:

- a1. Resource reservation requests Client-to-IDB (administrator or normal user or middleware).
- a2. Topology requests Client-to-IDB (administrator only).
- b1. Resource reservation requests to NRPS (normal operation).
- b2. Topology requests IDB-to-IDB within the NSP (topology exchange).
- b3. Resource reservation requests IDB-to-IDB within the NSP (topology exchange).
- c1. Topology requests HNA-to-IDB within the NSP (topology exchange).
- c2. Resource reservation requests HNA-to-IDB (request forwarding).
- d. NRPS-dependent interface.
- e. Network device dependent interface.

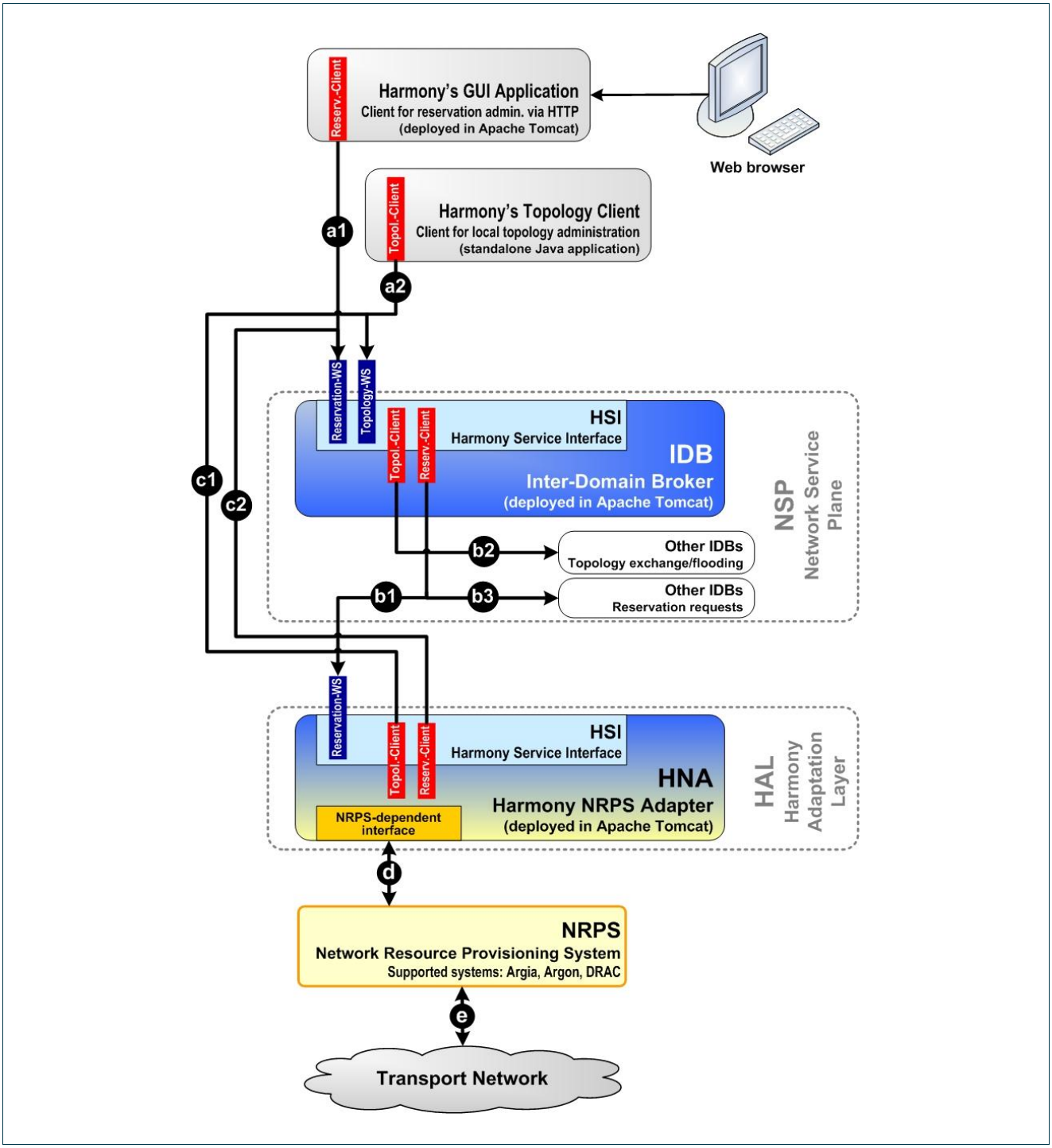

Figure 1.1: Simple Harmony system set up.

2 Distributing NSP/IDB functionality

The new prototype offers two modes for distributing the NSP functionality to several IDBs, the “hierarchical” mode and the “peer-to-peer” mode. These two modes can also be combined.

2.1 Hierarchical NSP

2.1.1 Basic functionality

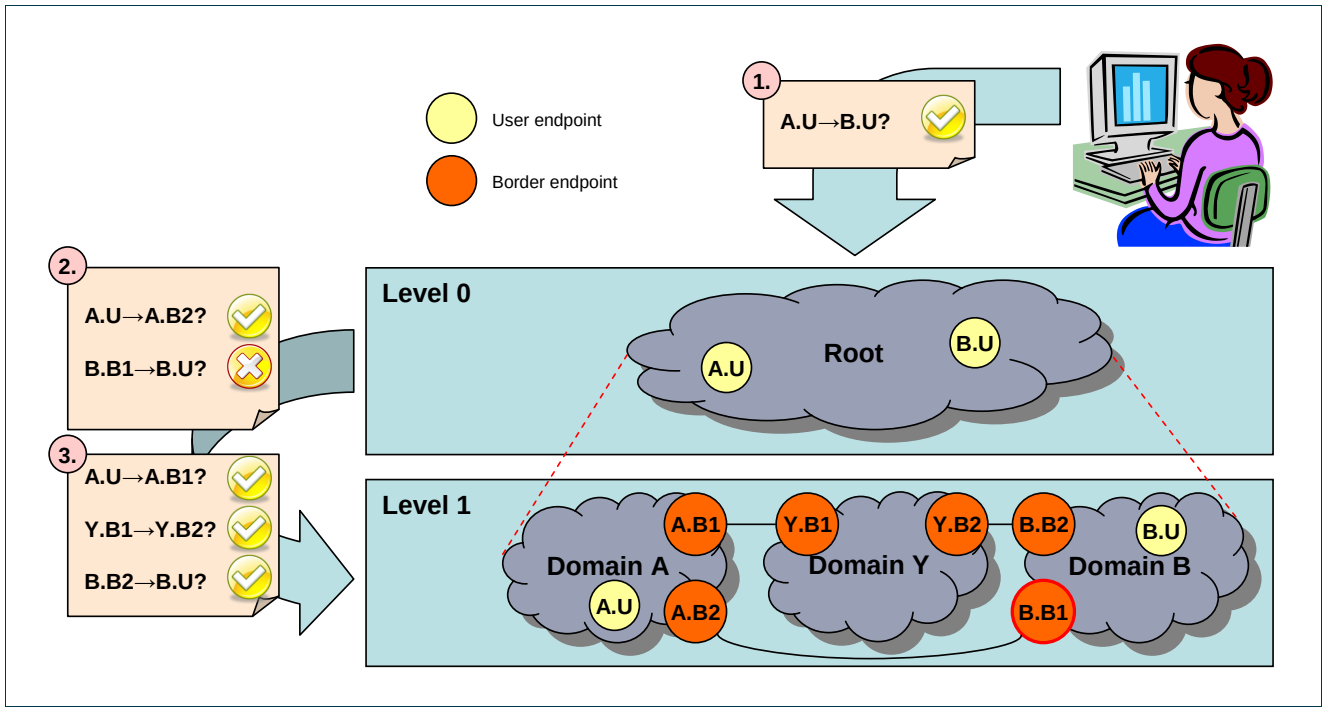

Figure 2.1: Simple hierarchical architecture (initial NSP design)

The NSP in its original design has already implemented the hierarchical mode in its most basic form: A single (root) IDB controls a number of domains controlled by HNAs (cf. Figure 2.1; for the terminology, cf. Section 1). In the general case, a central “root” IDB does not only control HNAs. Instead, IDBs can aggregate underlying domains to appear as a single domain towards higher level IDBs (cf. Figure 2.2). In the WP1 testbed, this mode of operation is used to aggregate the two HNAs used in the VIOLA testbed – the ARGON adapter and the “Thin NRPS” for GMPLS domains – to a single VIOLA domain towards the central IDB.

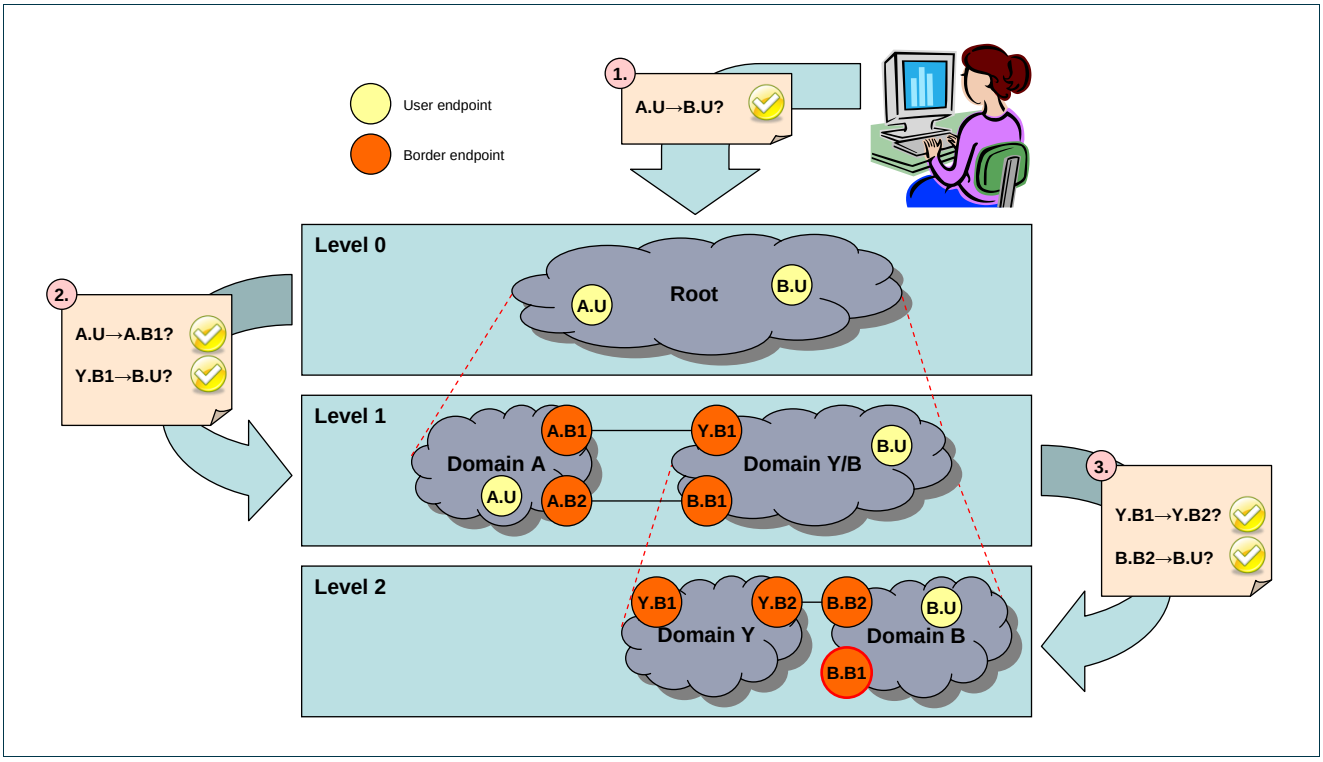

Figure 2.2: A more complex hierarchical architecture

In comparison to the peer-to-peer mode, the hierarchical mode of operation was rather simple to implement, because no changes to the previously existing interfaces were required.

2.1.2 Forwarding requests to parent IDB

A new feature implemented in the hierarchical mode is the "forwarding" of requests to parent IDBs if they contain unknown endpoints. Considering the example given in Figure 2.2, the request for the connection A.U→B.U need not necessarily be sent to the root IDB, but might as well be sent e.g. to domain A. The IDB controlling this domain would notice that it cannot locate the endpoint B.U and therefore forward the request to its parent IDB, which is the IDB controlling the root domain.

In order for this mechanism to work, the reservation ID semantics had to be modified. While previously, a reservation ID was merely a string that did not have any predefined semantics, an IDB now needs to know whether the reservation ID references a reservation in its own database or whether it has only forwarded this ID that it has received from its parent IDB. Therefore, the IDBs are required to add their domain names as suffixes for all reservation IDs they generate. Considering the previous example, a reservation reply for a reservation A.U→B.U would return an ID that ends in "@Root", regardless of which IDB served as entry point for the request. Now, if e.g. a cancel request is sent to the same IDB, it will know that it has to forward the request to the parent domain again, since the reservation ID does not end in its own domain ID.

2.2 Peer-to-peer NSP

The main novelty of the peer-to-peer mode towards the previous NSP implementation is that there is no central IDB anymore. Every IDB is aware of all peer IDBs and knows which endpoints they are in charge of.

2.3 Automatic Topology Exchange

The NSP interfaces were defined with a centralized system in mind, whose topology information is added and modified through an administrative interface, the Topology-WS. To reduce the difficulties of migrating to the enhanced system as much as possible, the changes to the already defined interfaces have been kept at a minimum. Actually, it was possible to implement the automatic topology exchange merely by adding optional fields to the XML schema definitions, so that e.g. the DRAC adapter, whose implementation was finalized prior to the main NSP enhancements, can still coexist with the enhanced NSP.

As service access point for update a domain's topology information the addDomain and editDomain functions in the Topology-WS have been identified. Since the prior is successful if and only if a domain is not yet known, and the latter is successful if and only if a domain is already known, an addOrEditDomain function has been added for more convenient topology exchange. This function must be implemented (additionally to the addDomain and editDomain functions) by an IDB that wants to participate in a peer-to-peer distributed NSP, not by HNAs.

The peer-to-peer topology exchange is implemented using only the addOrEditDomain function. Therefore, all relevant information is contained in this function's DomainInformationType parameter as optional fields. Legacy HNAs registering to an IDB can still use the addDomain or editDomain functions that also have the DomainInformationType as parameter.

The following fields have been added to the DomainInformationType:

- Sequence number: Numeric value allowing other domains to assess which domain information is "newer" than other domain information regarding the same domain.

- Interdomain link: An interdomain link is advertised as triple (destination domain name, link name, source TNA). The link name must be unique only with respect to the domains that are connected through this link, i.e. a link name can be "re-used" between different domain pairs. The combination of domain name and link name is used to identify the counterpart of an interdomain link in the domain information of the peer domain. An interdomain link can only be added to the topology database if it is actually "advertised" by both peers; otherwise, one of the TNAs is not known. This way of distributing the interdomain link information was chosen, because it only requires domain administrators to negotiate (abstract) link names, but does not require them to inform each other about specific TNAs used. E.g., in case of a hardware failure at a specific border endpoint, the domain administrator can

move the link endpoint to a different port (with a different TNA) and merely needs to change the domain information distributed by its own IDB and does not need to inform the administrator of the peer IDB.

- Relationship type: By setting the value of this field either to PEER or to SUBDOMAIN, a domain can indicate whether it is registering as a peer domain or as a subdomain.

3 Authentication and Authorization

The Harmony system has been provided with a security infrastructure, also called the Authentication and Authorization Infrastructure (AAI), which enables the possibility of performing secure cross-layer operations. In this case, cross-layer means from User/Middleware layer up to Network Resource Provisioning System layer, across the different layers defined in Harmony (cf. Section 1).

The AAI designed for Harmony is based on the OASIS WS-Security [WSS] standard and Message Level Security (MLS). This security model provides Harmony with the appropriate mechanisms for user identification –Authentication– and permission granting over a given (set of) resource(s) –Authorization.

The usage of WSS makes Harmony compliant with XML-Signature [xmldsig] and XML-Encryption [xmlenc] recommendations. The use of these two recommendations is driven by a Public Key Infrastructure model (PKI). The signature and encryption are used as in the typical PKI workflow, where the client signs the SOAP message with his private key and the server validates the signature with the client's public key. The following image illustrates this typical workflow.

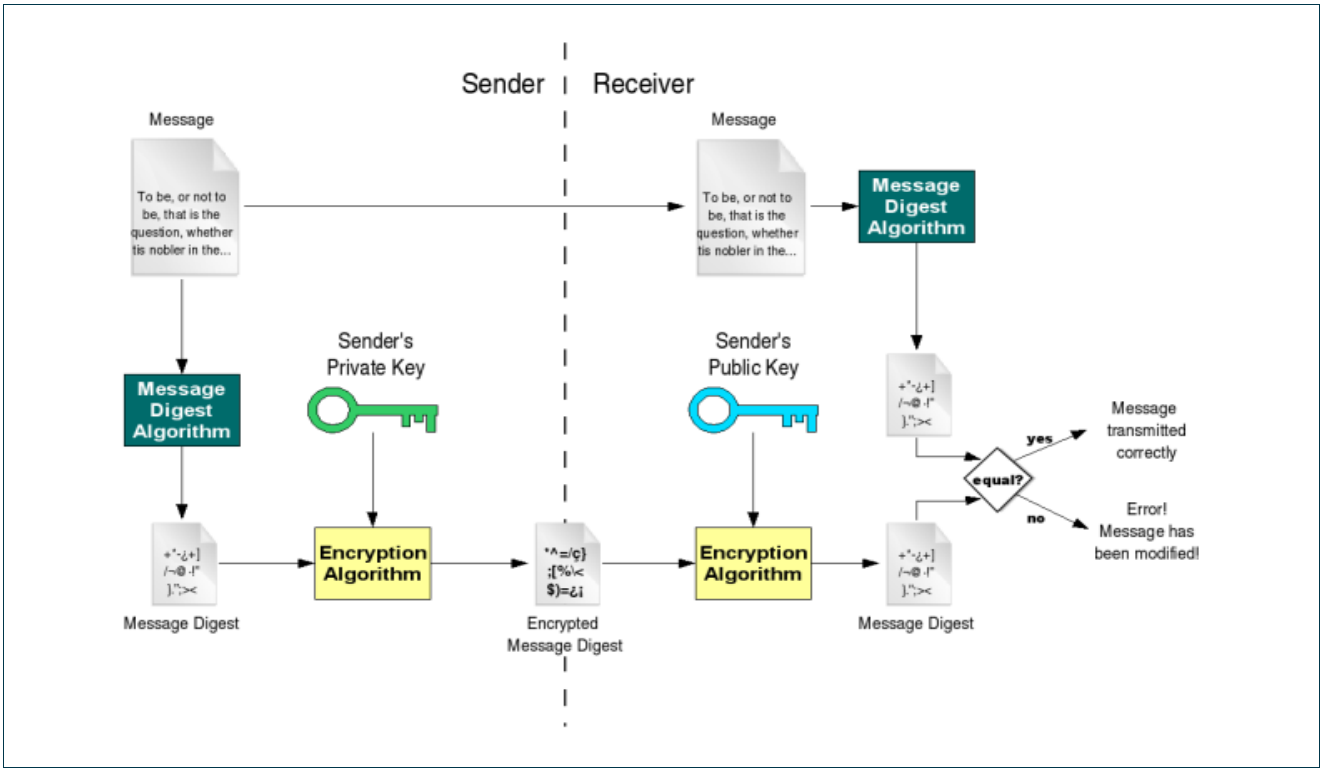

Figure 3.1: Typical PKI workflow. Sender is a client who signs a request message while Receiver is the server which validates and gathers the information required for a later authorization.

MLS is implemented by adding security information to the SOAP header of the service request messages. This information is added by using the suitable methods supplied by the Generalized AAA tool kit [GAAA-tk] libraries provided by WP4. In the following image, part of an *AAARequest* field can be seen where the user has added its *Id*, *role* and *context* for the service request and the given *Resource* he or she intends to use.

```
<soap:Header>
  <AAA:AAARequest xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
            xmlns:cnl="http://www.telin.nl/ns/#cnl"
            Id="CNLhashID#" version="2.0" cnl:attr1="CNL2test#">
    <AAA:Subject Id="WHO740@users.collaboratory.nl">
      <role>analyst@JobID;expert@JobID</role>
      <context>New application testing for Phosphorus</context>
    </AAA:Subject>
    <AAA:Resource>Resource element defines the target resource</AAA:Resource>
    […]
  […]
</soap:Header>
```

Figure 3.2: Example of security fields inserted in a signed SOAP header for a service request message in Harmony.

Harmony's AAI is constructed by two main modules:

1. The **Authentication (AuthN) module** is responsible for identifying the user and getting its information (mainly user identifier, role, and context). On the one hand, it adds/gets/checks user's signature and user's information to the SOAP header in the service request message. This information is structured in the form of SAML assertions [SAML]. On the other hand, AuthN module implements methods for SOAP body encryption/decryption which are based on Public Key Infrastructure (PKI) model and X.509 user certificates.

2. The **Authorization (AuthZ) module** checks/verifies which rules to apply and permissions to grant to an already authenticated user. The AuthZ module uses the user information for accessing a policy database and checking what policy/policies to apply for the given identity, role and context. Thus, it will permit or refuse a given user service request after authorization. The policy database uses XACML [XACML] language for describing the rules and actions to be taken.

In Figure 3.3, a block diagram of the components implemented in Harmony's AAI is depicted. Note that blue boxes are implemented specifically for Harmony and use the methods contained in the security library provided by WP4 (yellow box). Both certificate and policy databases have been designed by WP4 following WP1 requirements for Harmony. Both contain all the needed information for Harmony to authenticate and authorize service requests coming from the user (administration web application, Middleware, etc.).

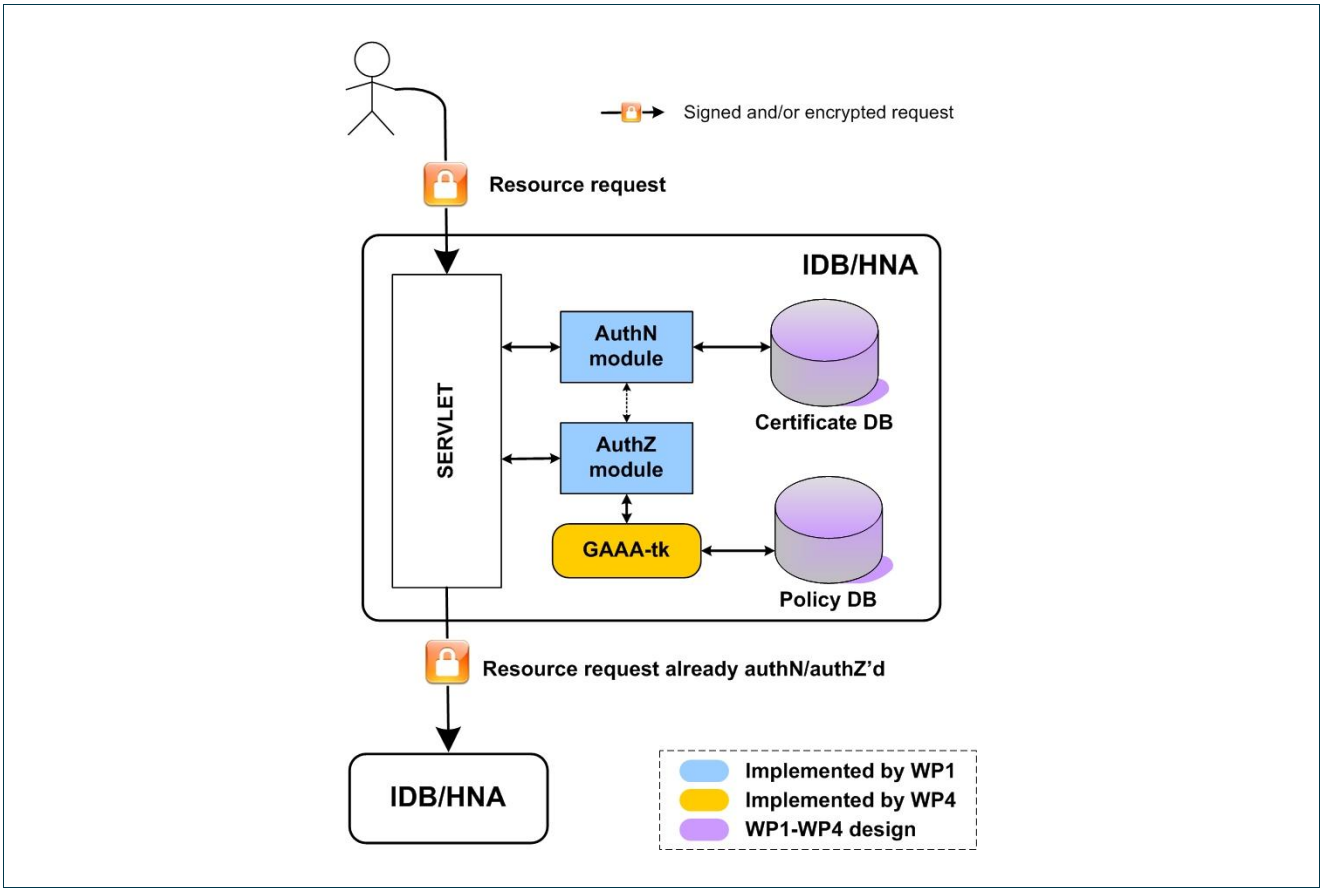

Figure 3.3: Block view of the security infrastructure (AAI) implemented in Harmony.

3.1.1 Security AAI – Details

3.1.1.1 *Java Packages involved in AAI*

The Security implementation is built in four main parts:

- **`harmony.common.security`**. This package implements the main modules and their methods for AuthN and AuthZ.
- **`harmony.common.serviceinterface`**. This package implements the security mechanisms in the client part of the HSI.
- **`harmony.common.serviceinterface.CommonServlet`**. This package implements the server part of the HSI with the security capabilities.

- **`harmony.ui.reservationcli`**. This part of the user interface implements the command line reservation client which also uses the security infrastructure implemented under `harmony.common.security`.

3.1.1.2 *Authentication implementation*

The implementation of the AuthN module uses SAML assertions in the service request message. SAML is an XML-based standard for exchanging authentication that incorporates different authentication statements such as signatures, user credentials or actions. These assertions are added in the SOAP message as set of tags inside the SOAP header that the client adds to its request.

The authentication model with SAML is implemented in Java [java-sun]. The SOAP assertion is created as a java object (using `org.w3c.dom`, it is included in the standard Java libraries). It is added as a set of different tags in the request.

For the signature and certificate part, the implementation uses the `org.apache.xml.security` [apache-xml-security]. This library provides a set of methods for the creation and management of keys or certificates for the security sub-system.

The implementation of the user certificate database uses a Keystore Java object (java.security.Keystore). The Keystore is a container for saving keys or certificates used in Java in secure communication environments.

The security information, as explained before, is added into the SOAP header of the service request message. Three main data blocks can be distinguished from the security information:

- User data
- Signature data
- User certificate and encryption data

User data

These fields contain all the necessary information to perform user identification and extraction of the context of the service request. The following table shows a detailed view of the different tags used, description and examples:

Tags	Description	Example
ResourceID (string)	The resource has the URI of the adapter the user wants to use and the	http://testbed.ist-phosphorus.eu/viola/harmony?source=1.1.1.1&target=2.2.2.2

	source and destination endpoints of the reservation *ResourceID = URI_adapter + SourceEp + DestinationEp*	
Action (string)	Operation invoked by the user	create-path, cancel-path, get-status, …
SubjectMap (HashMap)	Contains the Subject parameters (subjectID, subjectRole and sugectContext) described below	[test-user@localhost, student, demonstration]
subjectID (string)	User identifier	test-user@localhost
subjectRole (string)	User role(s)	Admin, student, professor, …
subjectContext (string)	Context of the service request	Demo, application-testing, …

Table 3.1: Harmony tags for user data exchange using SAML assertions.

Signature data

All the needed data for the signature is contained in these fields. They are XML-Signature compliant [xmldsig].

User certificate and encryption data

All the needed data for encryption is contained in these fields. They are XML-Encryption compliant [xmlenc].

The following image shows a full, signed SOAP header in a create reservation request message.

```
<soap:Header>
  <AAA:AAARequest xmlns:AAA="http://www.aaauthreach.org/ns/#AAA" xmlns:cnl="http://www.telin.nl/ns/#cnl"
            Id="CNLhashID#" version="2.0" cnl:attr1="CNL2test#">
    <AAA:Subject Id="WHO740@users.collaboratory.nl">
      <role>analyst@JobID;expert@JobID</role>
      <context>demo001</context>
    </AAA:Subject>
    <AAA:Resource>Resource element defines the target resource</AAA:Resource>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
        </ds:CanonicalizationMethod>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></ds:SignatureMethod>
        <ds:Reference URI="">
          <ds:Transforms>
```

```
                <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></ds:Transform>
                <ds:Transform
                   Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"></ds:Transform>
             </ds:Transforms>
             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
             <ds:DigestValue>9dd75eK2AFA4ZTLHqb0NTmyQKCA=</ds:DigestValue>
          </ds:Reference>
       </ds:SignedInfo>
       <ds:SignatureValue>
          PzZw2H3+kFxVfr99cspADgjcWBjD2U7VDKao4feD0KpqZIZpzk09deGTiXe2asgTIiO3MZrHxwv8
          ULPIJspWPbd8lZGp0BnEuwLoF1TymazH0VjsN3wFFKsUEFR4Ynjlq50nJzQzGIJzuDT+V6+aX4Wf
          GVoKBa8as4rx3wStR34=
       </ds:SignatureValue>
       <ds:KeyInfo>
          <ds:X509Data>
             <ds:X509Certificate>
                MIIB3jCCAUcCBEIGZ58wDQYJKoZIhvcNAQEEBQAwNjELMAkGA1UEBhMCTkwxGTAXBgNVBAoTEENv
                bGxhYm9yYXRvcnkubmwxDDAKBgNVBAMTA1BFUDAeFw0wNTAyMDYxODUzMTlaFw0wNTA1MDcxODUz
                MTlaMDYxCzAJBgNVBAYTAk5MMRkwFwYDVQQKExBDb2xsYWJvcmF0b3J5Lm5sMQwwCgYDVQQDEwNQ
                RVAwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANaYCBpYVu/6Ath11/wQ7r/gqd7K+h+gtS8X
                e/KhibziMna6PTLpfUInc0z9UlJVfv2+/E0q/bHb1RYbyNHZB4TcjrOQfQTKzWeeBwSfr2TGVHcu
                8PdggbL9NzXhrD9pGzoeSy62MUb3JxrQ0dFpNrq/8dDxeKO4cB4LuThH8S2TAgMBAAEwDQYJKoZI
                hvcNAQEEBQADgYEAvw28KeGVGKzv1M3gZmah5NBzsHLAkUpjKEGZGwKDjuvCF/PMiqNdobipP49i
                Af2ZnB0f+DoaoENT4+K46MsIJPYZTCzwVOyA0FttoWVvmyUAbcAA3byk8BYF9L3ajyHbcb30Dtm1
                9/i/9mAu5eKXK+nZucsVNIxdLfwaHKrxhfE=
             </ds:X509Certificate>
          </ds:X509Data>
          <ds:KeyValue>
             <ds:RSAKeyValue>
                <ds:Modulus>
                   1pgIGlhW7/oC2HXX/BDuv+Cp3sr6H6C1Lxd78qGJvOIydro9Mul9QidzTP1SUlV+/b78TSr9sdvV
                   FhvI0dkHhNyOs5B9BMrNZ54HBJ+vZMZUdy7w92CBsv03NeGsP2kbOh5LLrYxRvcnGtDR0Wk2ur/x
                   0PF4o7hwHgu5OEfxLZM=
                </ds:Modulus>
                <ds:Exponent>
                   AQAB
                </ds:Exponent>
             </ds:RSAKeyValue>
          </ds:KeyValue>
       </ds:KeyInfo>
    </ds:Signature>
  </AAA:AAARequest>
</soap:Header>

<soap:Body>
[…]
</soap:Body>
```

Figure 3.4: SOAP message of a client request

3.1.1.3 *Authorization implementation*

The authorization module works with the GAAAPI library provided by WP4. This library provides different tools for the implementation to a policy system, and tokens/tickets for a same context work, these tools will be used for the authorization process,which checks the polities and it permits storing a session context.

In the workflow, when the authentication is finished, the authorization gets the request information and checks the user permissions with a policy DB that contains a list of policies and describes the access permitted for each request.

For the policy specification, the authorization uses XACML. It is a declarative access control policy language implemented in XML and a processing model, describing how to interpret the policies The XACML sentence defines the correct access to some resource. Each sentence uses different parameters for its specification (subjectId, subjectRole, subjectContext, action and ResourceId).

4 Multi-Constraint Support

4.1 Bandwidth Management

In the EndpointType data type that is used in the Harmony interface, the field Bandwidth of type integer is present. The Bandwidth represents the Endpoint's bandwidth in megabits per second. In the first implementation of the Harmony IDB we have only administrated whether an interdomain link is in use. To support bandwidth management in case an interdomain link can be shared over multiple connections (which is not the case in the current implementation of the testbed) we decided to administrate the amount of megabits per second in use.

4.2 Multi-Technology support

We have included an optional Technology field of type EndpointTechnologyType (see Figure 4.1) in the EndpointType. The EndpointTechnologyType is a complex type consisting one or more EndpointSupportedAdaptation. An EndpointSupportedAdaptation is a string, of which the value is an URI to an NDL description of an Adaptation that is supported by the endpoint. All EndpointSupportedAdaptations together exactly define the technology of an endpoint.

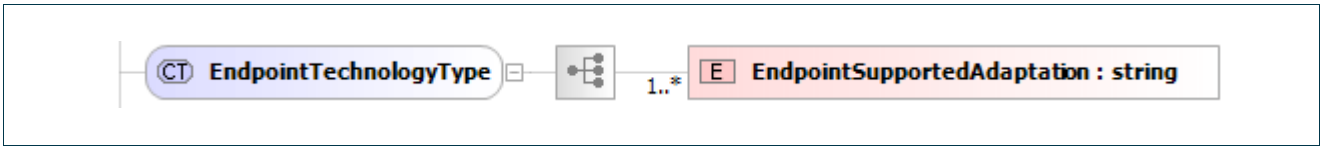

Figure 4.1: The EndpointTechnologyType

Also, we have included an optional Technology field of type DomainTechnologyType in the DomainInformationType (cf. Figure 4.2). The DomainTechnologyType is a complex type consisting of zero or more DomainSupportedAdaptation, zero or more DomainSupportedBandwidth and one or more DomainSupportedSwitchMatrix. A DomainSupportedAdaptation is a string of which the value is a URI to an NDL description of an adaptation that is supported by the domain. The DomainSupportedBandwidth is of type long and represents a bandwidth (in megabits per second) that is supported by the switching matrix of the domain. The DomainSupportedSwitchMatrix is a string of which the value is a URI to an NDL description of a switching matrix. The DomainSupportedSwitchMatrix defines the technology that a domain can use to make switching decisions (e.g. Ethernet, Sonet, IP).

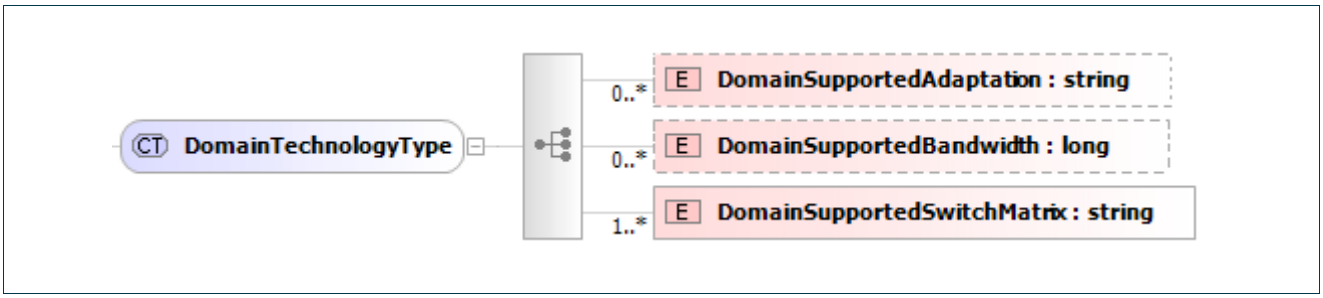

Figure 4.2: The DomainThechnologyType

4.3 Abstract Domain Topology

The definitions in Section 4.2 define a domain's capabilities in a general and abstract way. We have chosen not to specify the inner domain topology. A domain is rather viewed as a big abstract switching device with certain capabilities. We assume that a domain is fully interconnected, which may not always be the case. As a result of this choice, a domain cannot expose that a path between two endpoints in this domain does not exist. On the other hand the technology types can unveil that a path between two endpoints can never be valid and thus can be discarded.

5 Malleable reservations

Besides the commonly used fixed reservations, the IDB now supports malleable reservations, which are an extension or generalization of the fixed reservations. The users' intention for such a reservation is to transfer a certain amount of data within a given deadline. Additionally, constraints such as the minimum and maximum bandwidth supported by the application have to be considered. Ultimately, the user is only interested when the data transfer will start and how long it will take.

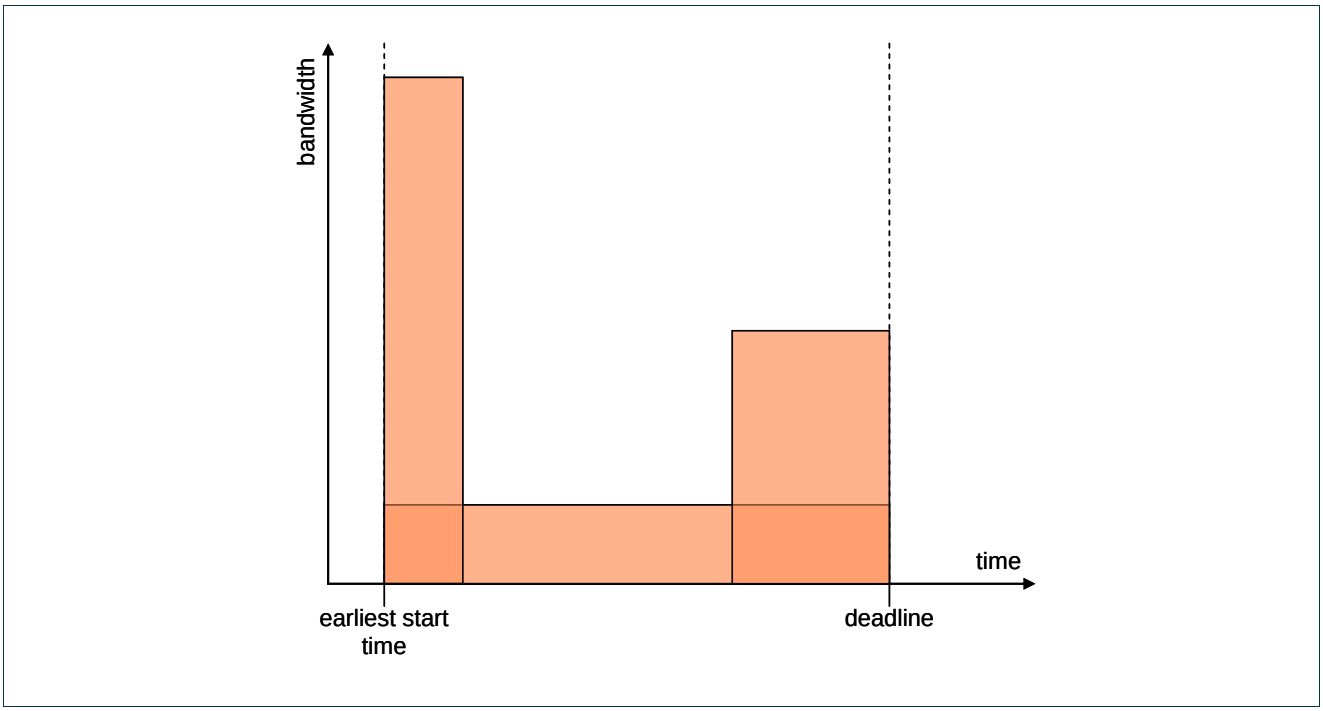

Figure 5.1: Three different configurations of a malleable reservation request

Figure 5.1 shows an example for three different configuration of a malleable reservation. For all three, the same amount of data (i.e. the product of bandwidth and duration, represented by the area within a rectangle) is transferred.

5.1 Malleable reservation setup

When a request for a malleable service is received, the IDB loops over different interdomain paths, possible start times and bandwidths to find a set of available resources that fulfil the original request. The different steps that are executed are described in the following subsections.

5.1.1 Path computation and choice of bandwidth

In a first step, a path between the given user endpoints is computed. For all endpoints on the chosen interdomain path, the feasible bandwidths are retrieved from the database. Only bandwidths which are in the range of the minimum and maximum bandwidths provided in the request are used for the further computation. If none of the database bandwidths fulfils the range condition, the maximum bandwidth given in the request is used. Upon failure to create a reservation, the bandwidth is halved until the minimum bandwidth given in the request is reached. If necessary, the same procedure is repeated for alternative paths. If no feasible path can be found, a PathNotFoundFault is sent back to the requestor.

5.1.2 Bandwidth adaptation

Now the attempt is made to adjust all endpoints on the given path to a common bandwidth level. At the beginning, for each endpoint the greatest feasible bandwidth is chosen, and among these, the minimum (the so called *min-max bandwidth*) is taken. In the second step, the highest bandwidths are reduced according to the technology-induced granularity until the smallest bandwidth which is equal to or greater than the min-max bandwidth is found. This way, the highest feasible bandwidth is selected for all endpoints on the given path.

5.1.3 Availability checking

In this step the availability of the network resources is queried at the appropriate IDCs with the previously evaluated constraints (endpoints with according bandwidth, duration and a now fixed start time). The possible results are:

- AVAILABLE: A reservation is feasible in this IDC.
- ENDPOINT_NOT_AVAILABLE: One or more endpoints are blocked for this duration. An alternative start time offset should be returned, which can be used for start time adaptation.
- PATH_NOT_AVAILABLE: An intradomain path is blocked for this duration. An alternative start time offset should be returned, which can be used for start time adaptation.
- AVAILABILITY_NOT_CHECKED: No availability check has been performed (should only be returned if other connections in the same reservation requests have already been reported as unavailable).

If one or more of the queried IDCs return a result different to AVAILABLE, the following adaptation attempts are carried out:

1. Start time adaptation: The start time for the reservation is shifted to the maximum of all received alternative start time offsets. If no offsets have been sent back, a default offset of one hour is used.

After this, the availability checking will be repeated with the new start time. If the new start time and the duration exceed the user-defined deadline, attempt 2 is carried out.

2. Bandwidth adaptation: Due to the strategy of starting with the maximal feasible bandwidth, it is now possible to decrease the bandwidths of the participating endpoints for one level. This yields a greater data transfer duration on the same path, but on this new bandwidth levels the endpoints or intradomain paths may be available again. If this adaptation is successful, a new availability checking loop is started with the original user-defined start time and the new bandwidths on the same path as before. If this attempt should also fail, all endpoints reported as blocked are removed from the set of feasible endpoints for path computation, and a new path computation is started (section 5.1.1).

5.1.4 Reservation

In this final step, the fixed reservations that have been verified to be available at the involved IDCs are committed to the IDCs. The resulting fixed interdomain representation fulfils the original malleable reservation constraints.

6 Notification framework

Originally, it was planned to use the WS-Notification framework that is part of the WSRF. However, it has turned out that the existing WSN implementation does not meet the requirements within the Network Service Plane. The reason is that notifications within the NSP mainly concern specific reservations, i.e. the topics are given by the existing reservations. Therefore, it is necessary to create new topics on the fly, one for each new reservation. After some analysis of and experimenting with the existing WSN implementation, it has been decided that the risk is too high that enhancing it to fulfil this requirement might induce a greater effort than creating a new, very basic notification framework implementation tailored specifically for the NSP.

Therefore, a new notification framework has been implemented which is very similar to the WSN framework. The roles of the participating entities are those of *notification producers* and *notification consumers*. A notification consumer merely implements a *notification* operation in its Reservation-WS. The parameters of this operation are the *topic* (i.e. a reservation ID) and a list of *ServiceStatusTypes* (see [Phosphorus-D1.4]) that inform the consumer of the new status of one or more of the services that belong to the reservation ID. This basic mechanism eliminates the need to periodically poll systems to see whether a status has changed.

The administration of topics and subscriptions is handled by a separate *Notification-WS*. A consumer wishing to subscribe for notifications on a certain reservation simply calls the *subscribe* operation with a topic that is constructed from the reservation ID and the domain ID that assigned the reservation ID. While it would be sufficient for the current architecture to use solely the reservation ID as topic name, this approach was taken to allow a single Notification-WS to be in charge for several domains. A subscription can be cancelled with the *unsubscribe* operation.

7 Further enhancements

7.1 Cached communication and locking concurrent reservation requests

7.1.1 Cached communication between IDB and NRPS

When coordinating multidomain reservations between heterogeneous resources, the IDB typically has to forward multiple requests related to a single multidomain reservation to different NRPSs. But sometimes, an IDB entity within the NSP has to forward the requests to the same domain more than once, asking for the availability of the same network resources as the previous request. In order to avoid multiple requests asking for the same resources and the resulting signalling overhead, a cache between the IDB and the HNAs has been implemented.

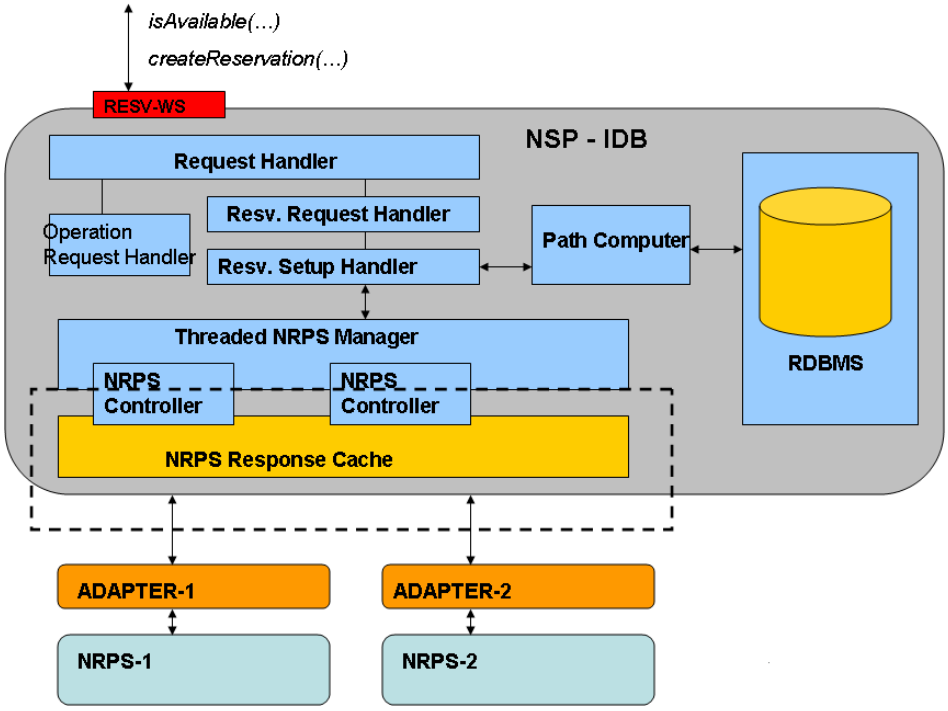

Figure 7.1: Internals of the Inter-Domain Broker entity in the NSP. The dotted area shows the position of the NRPS Response Cache and its interaction with the NRPS Controllers and the Harmony NRPS Adapters.

The NRPSResponseCache stores the responses from the underlying domains for a short period of time. This period of time is configurable by means of changing the flag cache entry timeout in the interdomain broker properties file. When the timeout expires, the cache entry is removed automatically.

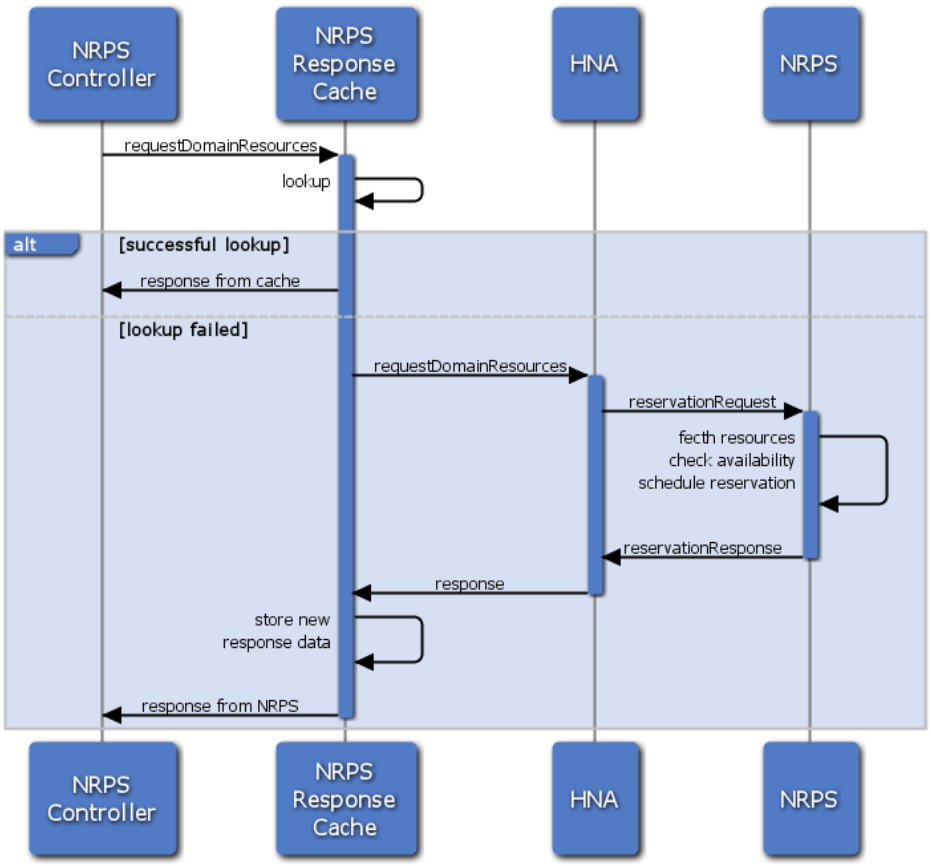

Figure 7.2: Sequence diagram showing the two functional alternatives the NRPS Response Cache offers: successful or failed lookups.

A cache entry contains information about the availability of a path with a given constraints inside a domain. It is, in essence, a *connectionConstraintType* and the *alternativeStartTime*. The NRPS Response Cache is a singleton class. The methods defined in the cache are the three basic methods regarding a cache-like behaviour: *addEntry*, *removeEntry* and *lookup*. The first method inserts in the cache all the connections provided in the *isAvailableType* object with their corresponding responses. It also programs the deletion of the connections after a given interval time. The second method removes all the information stored in cache automatically while the last method looks in the cache for a response.

7.1.2 Locking concurrent reservations request

When coordinating multi-domain advanced resource reservations, the entities in the service plane (IDBs) could receive multiple concurrent incoming requests. This might lead to situations where several requests, if processed sequentially, could all be fulfilled, but are denied when processed in parallel, because they block each others resources. To prevent such situations, an exclusive create reservation mode has been implemented. This mode can be enabled in the corresponding properties file of an IDB. In case this exclusive create reservation flag is enabled, an IDB blocks all incoming reservation requests while one createReservation is processed.

7.2 Parallelized communication

The module of the NSP in charge of the communication with the subdomains or NRPSs is the NRPS Manager. This module is invoked internally by other blocks of the NSP when one or more messages have to be sent to one or more NRPSs. When the manager is invoked, it creates a proxy for each NRPS web service, sends the message and waits for the responses. Afterwards, it returns all responses to the invoker.

The responses received in reply to these requests have to be analysed to take further action or to construct a response in reply to a request received on the reservation interface of the NSP. However, some types of replies require immediate action. If a single request in a series of CreateReservation fails, then a rollback is required. Hence, the reservations that have already been established in other domains must be cancelled.

The requests to the NRPS manager are composed by tuples <Domain, MessageToSend>, and the responses consist of pairs <Domain, ResponseReceived>. All these responses coming from each NRPS adapter are returned to the invoking request handler. In the case that the overall reservation process has not been successful, the NRPS manager will create the necessary CancelReservation requests to domains that have reported a successful reservation creation.

As depicted in previous deliverables, the second version of this NRPS manager is a threaded version, not sequential. This has been developed with the idea of avoiding large waiting times by sending the requests to all the NRPSs in parallel. This way of sending the messages allows all the NRPS to receive the request nearly at the same time and they can work concurrently, shortening the overall response time of the system.

The NRPS Controller has been developed as a thread extending class. It contains all the information required to contact the NRPS, since it is used to communicate with an instance of an adapter/NRPS. Inside the components of this class, we would like to mention the proxy for the reservation web service, the endpoint references to the topology and reservation web services, the operation to send and the message to send. One NRPS Controller thread is created when one request has to be sent to the NRPS. The NRPS Controller information is filled and the thread starts to work in a parallel way.

7.3 DNS-Lookup

When creating multidomain reservations, the user has to identify the involving source and target endpoints. As described in Section 1.2 of [Phosphorus-D1.6] the Service Plane operates on OSI Layer 2 and hence identifies the endpoints by so called TNAs, identifiers that follow the IPv4 syntax. Normally, the end user who requests a path is only aware of the involved IP addresses or FQDNs, but does not know the corresponding TNA addresses. In the past the user was forced to look on a map that mirrors the current testbed in order to map between TNAs and IPs.

The newly implemented lookup technique had to meet the following requirements:

1. Do not change the internal communication
2. Resolve both IP addresses and host names into TNAs
3. Find a scalable solution

Therefore, the WP1 client library (used by the GUI, the CLI, and other implementations) was extended to resolve given endpoint identifiers (name or IP) to valid TNA addresses by using TXT DNS records. E.g.:

```
zam668.viola-testbed.de.                    IN     TXT TNA=10.7.12.108
72.0.0.10.ip-lookup.viola-testbed.eu.       IN     TXT TNA=10.7.12.108
```

Code 4.4.1: Example DNS records

```
$ nslookup -q=txt zam668.viola-testbed.de
Server:     131.220.6.18
Address:    131.220.6.18#53

Non-authoritative answer:
zam668.viola-testbed.de text = "TNA=10.7.12.108"
```

Code 4.4.2: DNS lookup example

Based on these DNS records the following algorithm tries to find the corresponding TNA:

```
1 $tna = getTNA($nameOrIP);
2 if (!$tna && !isIPorTNA($nameOrIP)) {
3     $domains = {.viola-testbed.de, .i2cat.viola-testbed.de, ...}
4     foreach ($domain in $domains) {
5       if (!$tna) $tna = getTNA($nameOrIP + $domain));
6    }
7 }
8 return $tna;
```

Code 4.4.3: DNS resolving algorithm

Using this approach the following user input can be resolved to TNA 10.7.12.108 (see Code 4.4.3):

- Line 1: 10.0.0.72 (input: IP)
- Line 1: zam668.viola-testbed.de (input: FQDN)
- Line 2: 10.7.12.108 (input: TNA)
- Line 5: zam668 (input: name)

Important in this context is the fact that a single IP or FQDN could be bound to more than one TNA address. In the current implementation the first TNA found will be returned. If the user wants to use a specific port, he still needs to identify it using the corresponding TNA. Also, since this solution is implemented on the client side only, it is not a feature of the Harmony Service Interface itself.

8 Conclusions and Outlook

In the second phase of the Phosphorus project, a number of enhancements have been added that increase the stability, usability and performance of the system. In this document, we have described these enhancements in the prototype code that is delivered.

While in the face of the rather short development time, great achievements have been made regarding the NSP implementation, some key issues were not addressable in this time:

- Point-to-Multipoint: While the interface was designed to be point-to-multipoint capable, the core code assumes that there is only a single destination endpoint at several places. Adding the point-to-multipoint capability here would be a complex task.

- Modification Request: Some underlying technologies allow the seamless changing of the reserved bandwidth on a given port. Including this facility into the Service Plane Interface would allow to dynamically modify the reserved bandwidth for a switched path. Using this request in particular in connection with Malleable Reservations could enhance the average utilization of a link.

- WS-Agreement: In a distributed service-oriented computing environment, service consumers like to obtain guarantees related to services they use, often related to quality of a service. Whether service providers can offer – and meet – guarantees usually depends on their resource situation at the requested time of service. The objective of the WS-Agreement specification is to define a language and a protocol for advertising the capabilities of providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime. Offering this interface would better meet the co-allocation requirements of Grid level scheduler to coordinate resource management systems located in different domains.

- Jobs: Support for grouping reservations to so-called “jobs”, allowing the middleware to cancel or commit to reservations that comprise a complex workflow is an interesting, but not essential feature. Therefore, it was put on hold in favour of other, more important enhancements.

- Translator: Interfacing other projects, like G-lambda, AutoBAHN, or IDC require the development of a bidirectional Gateway that translates reservation requests and handles the topology exchange. First experiences were made writing a Translator for the Inter Domain Controller and the first demonstration is planed for November 2008.

9 References

[GAAA-tk]	http://staff.science.uva.nl/~demch/projects/aaauthreach/index.html
[WSS]	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
[SAML]	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[XACML]	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[xmldsig]	http://www.w3.org/TR/xmldsig-core/
[xmlenc]	http://www.w3.org/TR/xmlenc-core/
[apache-xml-security]	http//santuario.apache.org
[java-sun]	http://www.sun.com/java/
[Phosphorus-D1.6]	Phosphorus Deliverable 1.6: “Planning and report on functional tests, and prototype for NRPS, Grid GMPLS control plane and middleware interoperability”.
[Phosphorus-D1.4]	Phosphorus Deliverable 1.4: “Definition and development of the Network servie Plane and northbound interfaces development”.

10 Acronyms

AAI	Authentication and Authorization Infrastructure
AuthN	Authentication
AuthZ	Authorization
GAAA-tk	Generalized Authentication, Authorization and Accounting Tool Kit
HAL	Harmony Adaptation Layer
HNA	Harmony NRPS Adapter
HSI	Harmony Service Interface
IDB	**Inter-Domain Broker**
MLS	Message Level Security
NRPS	Network Resource Provisioning System
NSP	Network Service Plane
RDBMS	Relational Database Management System
TNA	Transport Network Address
WSS	Web-Service Security