034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D1.5

Integration of the NSP and the Meta-Scheduling System (MSS) of the Service Layer within the middleware

Due date of deliverable: 2008-03-28
Actual submission date: 2008-03-28
Document code: Phosphorus-WP1-D1.5

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
Universiteit van Amsterdam

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

Abstract

This deliverable describes the integration of the Network Service Plane with the Service Layer of the middleware (the Meta-Scheduling System),

Project:	Phosphorus
Deliverable Number:	D1.5
Date of Issue:	03/28/08
EC Contract No.:	034115
Document Code:	Phosphorus-WP1-D1.5

Table of Contents

Table of Figures

Table of Tables

0 Executive Summary

This deliverable defines the integration of the Network Service Plane (NSP) with the Meta-Scheduling System (MSS). The main task of the NSP is to coordinate multiple network domains controlled by different administrative authorities such that this multidomain aspect is hidden towards the MSS.

1 Requirements for Network Service Plane and MSS integration

1.1 Requirements for the MSS

The Meta-Scheduling System (MSS) developed in WP3 is responsible for the co-allocation of Grid and network resources managed by different administrative entities. It is currently implemented in the UNICORE middleware, but it is designed as middleware independent as possible. The MSS does not distinguish between Grid and network resources. It treats the network in the same way as grid resource, whose availability can be queried and that can be reserved in the same way as processing power on a computing cluster. In a single domain environment, a Network Resource Provisioning System (NRPS) offers these services to the MSS. A NRPS is a system that has full knowledge about the underlying network's topology and the utilization of resources at different points in time.

For the interface between the MSS and the NSP the same technology as towards the Compute Resources has been selected at the side of the MSS: WS-Agreement. As the NSP itself has another interface we follow the same approach as we did in VIOLA with the local Resource Management Systems to keep the respective impact of the two components minimal: an adapter was implemented that wraps the thin reservation client of the Reservation-WS of the NSP. This adapter developed for KODAVIS provides WS-Agreement interface, that maps to the NSP reservation-WS functions. The interaction of the MSS and the NSP through this adapter, which is developed by WP3, is shown in Figure 1.1. The mapping of TNAs to grid resources was configured for the KODAVIS application on MSS side (no separate service or dynamic discovery is implemented yet). The mapping information was provided by WP1.

The current implementation of the MSS uses the `createReservation`, `getStatus` and `cancelReservation` of the NSP, but it does not yet include a preview functionality, which allows determination of the earliest start time for a requested service.

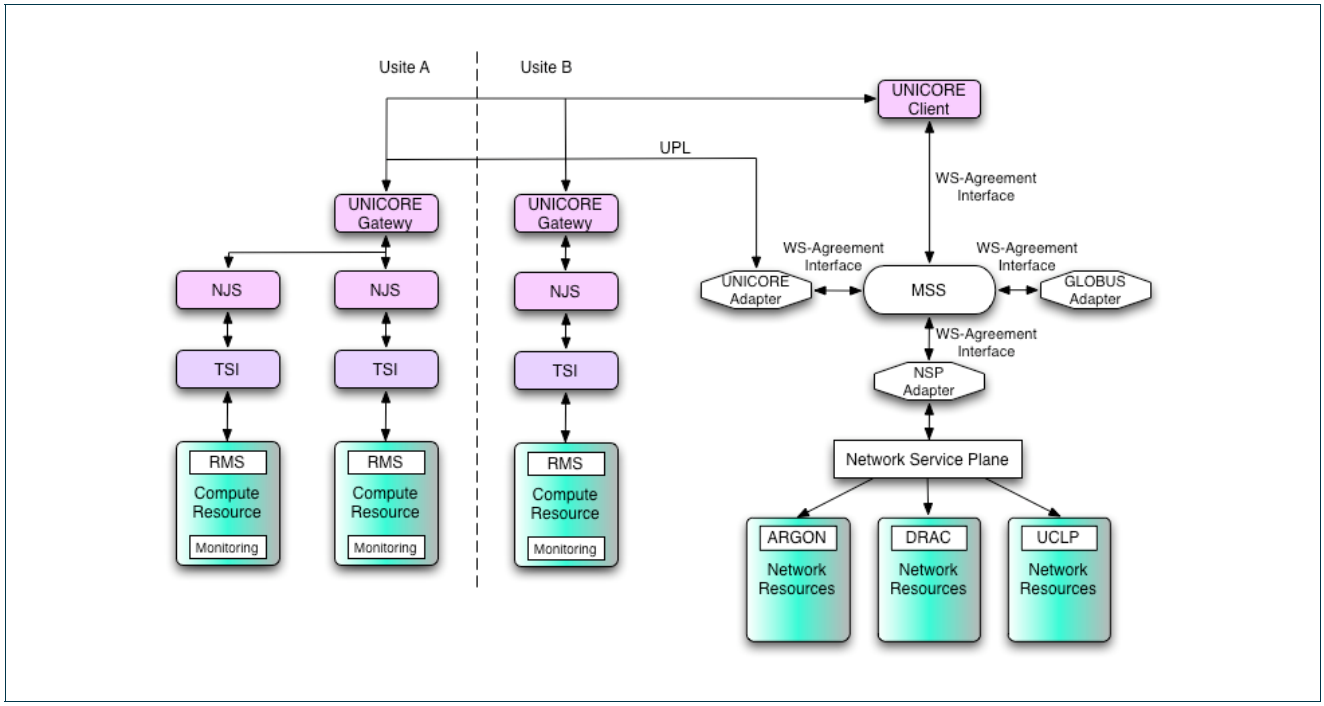

Figure 1.1: The MSS interacts with the NSP through the NSP adapter that maps the WS-Agreement interface of the MSS to the reservation-WS functions of the NSP

1.2 Requirements for the Network Service Plane

In a “multidomain environment” with several coexisting NRPSs, each of them controlling an own domain, one or more entities aware of the multidomain nature of the underlying network are needed to support a MSS. The interface such Network Service Plane (NSP) entities should offer is not essentially different to that offered by classical NRPSs; however, they operate quite differently. While a NRPS keeps track of the allocated resources and directly accesses the network devices to establish paths, a NSP entity acts as a mediator between one or more NRPSs and a MSS.

Because the actual resource management is implemented within a domain, it is not wise to duplicate the resource allocation schedules in the NSP entities. In certain cases, it may be useful to decrease response times by “caching” information inside the NSP, but care must be taken to avoid inconsistent information states. This is especially difficult since it cannot be expected that all resource requests pass the NSP: Local users will continue using their local NRPS interface to make local reservations. Note that the basic idea of NRPSs in contrast is that these systems solely control the bandwidth usage in the underlying network.

Therefore, in the basic implementation of the (single, central) NSP entity, no resource usage is recorded in the NSP, but every request is processed by coordinating the management systems of the underlying domains. The most complex operations are those that require a feasible interdomain path to be found. E.g., to make a

multidomain reservation, the NSP entity must find a set of intradomain connections that involve the requested user endpoints and several border endpoints that are interconnected by interdomain links.

Other operations basically are handled by duplicating the incoming request, adapting specific fields, and forwarding the requests to the involved domains. The replies must then be combined to a single reply in an appropriate way.

2 Network Service Plane Northbound interfaces

Integration of the MetaScheduler and the NSP is achieved through the northbound interface's operations. The northbound interface defines operations for management and setup of reservations, management of connections and retrieving of features of the NSP or the underlying domains. This way it facilitates co-allocation and utilization monitoring of resources at the MetaSchedular's level or any other client northbound to the NSP (**Figure 2.1**).

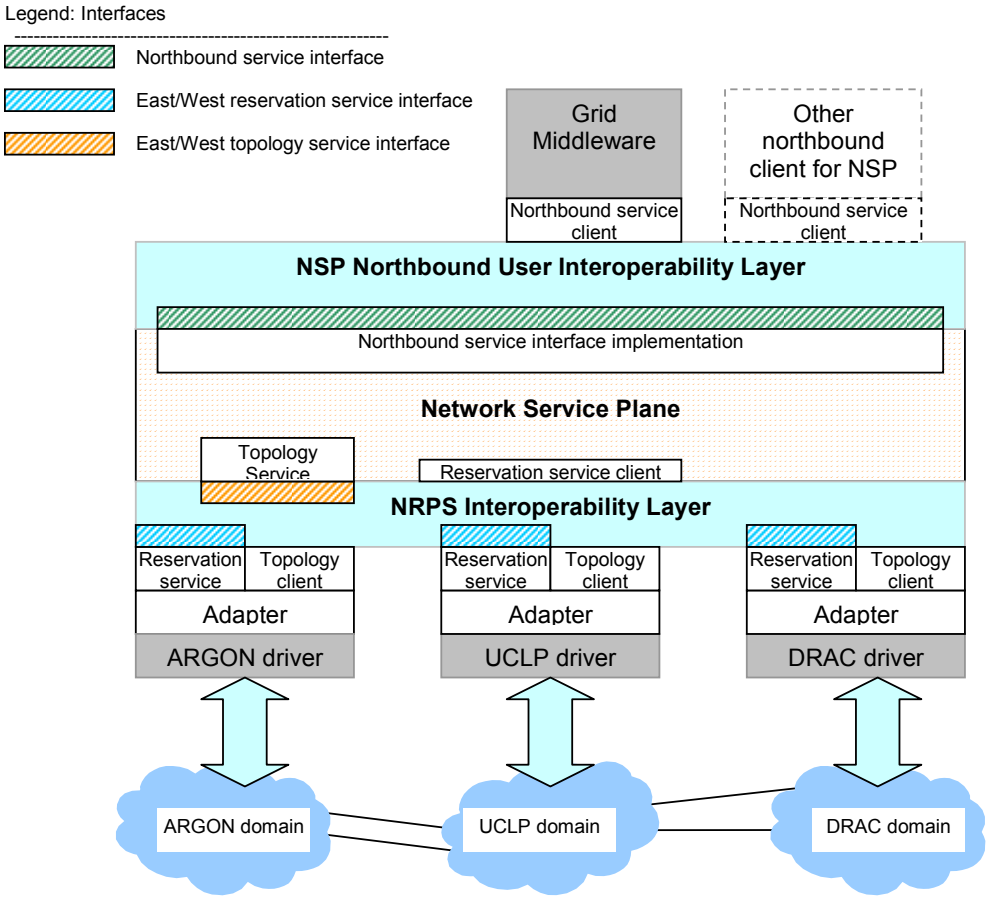

Figure 2.1: Overview over NSP interfaces and modules

Resources can be reserved, activated, bound and optionally cancelled through the `CreateReservation`, `Activate`, `Bind` and `CancelReservation` operations respectively. Resource utilization monitoring is facilitated through `GetReservation`, `GetReservations`, `GetStatus` and `IsAvailable` operations. Finally, features supported by the NSP can be requested using the `GetFeatures` method (see section 2.1). The northbound interface does contain additional operations but these are of less interest with regard to MetaSchedular integration.

Using the northbound interface operations requires providing operational constraints as arguments in the form of certain data types. The data types used on the northbound interface are introduced in Section 2.1.1.

2.1 Northbound interface

The reservation interface is essentially the same, with regard to resource provisioning, as the interface that allows for NRPS interoperability, i.e. the East/West interface which enable NRPS interoperability and NRPS integration into the NSP. This is the case because both the NSP and the NRPSs allow resource provisioning, however, the functionality provided by the reservation interface of the NSP is of a higher level coordinative nature. Typically, within the reservation interface the individual domain resource reservations are handled and presented to the higher level entities as one set of reservations.

In the following subsections, data types are defined as **data type name : (composite) data type**. Underlined (composite) types are defined in Section 2.1.1. Not underlined data types are XML schema defined data types. Composite data type which are not defined any further in the section are of less interest in the context of this document and are defined in detail in table 2.1 of the deliverable D1.4.

2.1.1 Reservation management

WSDL Operation Name	GetReservation
Description	Retrieves the input by which a reservation request was made
Input parameters (XML type)	ReservationIdentifierType : long ServiceIdentifierType : integer
Output parameters (XML type)	GetReservationResponse : GetReservationResponseType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	GetReservations
Description	Retrieves all existing reservations for the specified period of time
Input parameters (XML type)	PeriodStartTime : dateTime PeriodEndTime : dateTime
Output parameters (XML type)	*Sequence of* Reservation : GetReservationsComplexType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	CancelReservation

Description	Cancels a network resource reservation
Input parameters (XML type)	ReservationIdentifierType : long
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	**GetStatus**
Description	Returns the status of a service
Input parameters (XML type)	Service : ServiceConstraintType
Output parameters (XML type)	ServiceStatus : ServiceStatusType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

2.1.2 Reservation setup

WSDL Operation Name	**IsAvailable**
Description	Checks whether the specified service is available
Input parameters (XML type)	Service: ServiceConstraintType JobIdentifierType : long
Output parameters (XML type)	DetailedResult : ConnectionAvailabilityType *optional* AlternativeStartTimeOffset : long
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault
Fault (XML type)	EndpointNotFoundFault : EndpointNotFoundFault

WSDL Operation Name	**CreateReservation**
Description	Creates the reservation of a path between 2 endpoints considering the specified constraints
Input parameters (XML type)	Service : ServiceConstraintType NotificationConsumerURL : string

Output parameters (XML type)	ReservationIdentifierType : long Detailed result : ConnectionAvailabilityType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault
Fault (XML type)	EndpointNotFoundFault : EndpointNotFoundFault

2.1.3 Connection management

WSDL Operation Name	Activate
Description	Activates a service
Input parameters (XML type)	ReservationIdentifierType : long ServiceIdentifierType : integer
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	Bind
Description	Create binding between NRPS endpoint and application endpoint
Input parameters (XML type)	ReservationIdentifierType : long ServiceIdentifierType : integer ConnectionIdentifierType : integer EndpointID : EndpointIdentifierType
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

2.1.4 Other

WSDL Operation Name	GetFeatures
Description	Retrieves information about the supported features of the NSP / NRPS

Input parameters	None
Output parameters (XML type)	*sequence of* FeatureName : string
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

2.2 Data types

Constraints with regard to scheduling of resources are provided with the operations as arguments. Key types for defining the scheduling constraints are the `ServiceConstraintType` and the `ConnectionConstraintType` in which the type of service and the specifics of the connection can be set (see **Table 2.1**). A service can be either fixed, deferrable or malleable which can be defined through the `FixedReservationConstraintType`, `DeferrableReservationConstraintType` and `MalleableReservationConstraintType` dat types respectively. The bandwidth, delay and data amount are set within the `ConnectionConstraintType`. Specifics regarding the endpoints used in the connection and its directionality are specified in the `ConnectionType` which is the `ConnectionConstraintType's` base data type.

The service and connection constraints data types are also used in the operations for monitoring resource utilization. Additionally there are specific service and connection status information data types defining current statuses of resources. For services this is the `ServiceStatusType` and for connections the `ConnectionStatusType` both containing status information defined using the StatusType data type.

In the table below (**Table 2.1**) the above mentioned data types and there composition are described. The data types are defined as **data type name : (composite) data type**. Underlined types are composite types and either defined in the same table or, when less interesting in the context of this document, in table 2.1 of the deliverable D1.4. Not underlined data types are XML schema defined data types.

Constraint data types for scheduling and monitoring utilization of resources		
Data type name	**Description**	**(composite) Type**
ServiceConstraintType	Type used to specify constraints for a service	ServiceIdentifierType : integer TypeOfReservation: ReservationType *one of* FixedReservationConstraints : FixedReservationConstraintType DeferrableReservationConstraints : DeferrableReservationConstraintType

		MalleableReservationConstraints : MalleableReservationConstraintType AutomaticActivation : boolean *sequence of* Connections : ConnectionConstraintType
ConnectionConstraintType	Type of constraint on the connection	*extends* ConnectionType MinBW : integer MaxBW : integer MaxDelay : integer DataAmount : long
ConnectionType	Type of the connection	ConnectionIdentifierType : integer Source : EndpointType Target : EndpointType Directionality : integer * *Possible values: 0="unidirectional tree", 1="bidirectional tree", 3="full mesh"*
FixedReservationConstraintType	Constraints for fixed reservations	StartTime : dateTime * *Indicates the time when the service should start* Duration : integer * *Indicates the duration of the service in seconds*
DeferrableReservationConstraintType	Constraints for deferrable reservations	StartTime : dateTime * *The earliest point in time when the connection would be useful* Duration : integer * *Indicates the duration of the service in seconds* Deadline : dateTime * *The latest point in time when the service will be useful*
MalleableReservationConstraintType		StartTime : dateTime * *The earliest point in time when the connection would be useful* Deadline : dateTime * *The latest point in time when the service will be useful*
ConnectionAvailabilityType	Availability of the connection	ServiceIdentifierType : string ConnectionIdentifierType : integer Availability : string * *Allowed values:* (Enumeration) 'available' 'endpoint_not_available' 'path_not_available' 'availability_not_checked' *optional sequence of* BlockedEndpoints : EndpointIdentifierType MaxBW : integer * *Maximum available bandwidth in Mbps (only set if the corresponding MaxBW was set in the availability request)*

ServiceStatusType	Type for the service status	ServiceIdentifierType : string Status : StatusType DomainStatus : DomainStatusType Connections : ConnectionStatusType
ConnectionStatusType	Type of the status of the connection	*extends* ConnectionType Status : StatusType DomainStatus : DomainStatusType ActualBW : integer * *Actual bandwidth in Mbps*
StatusType	Type of the status	*one of (Enumeration)* unknown : string pending : string active : string completed : string cancelled_by_user : string cancelled_by_system : string setup_in_progress : string teardown_in_progress : string

Table 2.1: Data types used on the northbound interface. Composite data types that are not defined any further in this table are defined in table 2.1. of the deliverable D1.4

3 Implications of Network Service Plane architecture to the MSS

3.1 Architecture overview

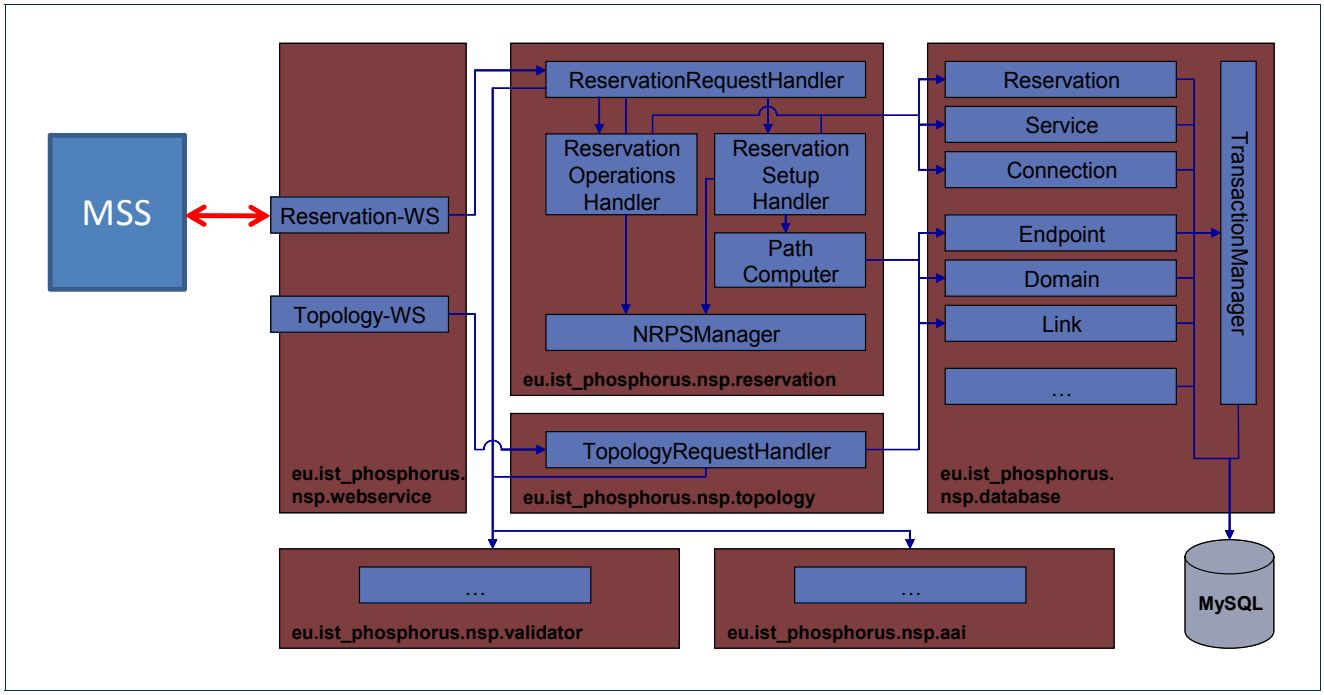

Figure 3.1: NSP architecture (simplified)

Figure 3.1 depicts a simplified overview of the NSP architecture, the dependencies between the different internal modules and the interoperation with the MSS. Requests from the middleware or from a user application are received through the webservice classes in the package *eu.ist_phosphorus.nsp.webservice*. These auto-generated classes contain very little functionality and mainly hand the received requests to a *CommonRequestHandler* class (not depicted in the figure) that validates the requests using the *eu.ist_phosphorus.nsp.validator* package (cf. Section 3.2.3), that authorize the requests using the

eu.ist_phosphorus.nsp.aai package (cf. Section 3.3), and that finally forwards the requests to the appropriate handlers.

The Java classes that represent the data communicated through the webservice interfaces are auto-generated from the WSDL specifications using JAXB (Java Architecture for XML Binding, [JAXB], cf. Section 3.2.1). Data to be stored persistently is represented through own Java classes in the *eu.ist_phosphorus.nsp.database* packag. These classes offer all functionality for database access, so it is not necessary that classes outside of this package are aware of details of the database used.

Requests received through the Topology-WS are processed by the *TopologyRequestHandler* (cf. Section 3.1.1), whereas requests received through the Reservation-WS are processed by the *ReservationRequestHandler* (cf. Section 3.1.2). This possibly results in communication between NSP modules and different NRPS adapters, which is encapsulated by the *NRPSManager*.

The MSS interacts with the NSP through the Reservation-WS to manage the required reservations. This interaction will be further detailed on the coming sections.

3.1.1 Topology webservice

All requests received through the Topology-WS are processed by the *TopologyRequestHandler* class. Basically, this class “translates” between the JAXB classes used on the webservice interface and the database classes used in the NSP core and vice versa. The MSS will not require the operations of this web service interface; this will only be used by the topology-aware clients like the NRPSs and the graphical topology client. Though, the MSS could invoke operations to this interface in the same way that it does with the Reservation-WS if it were necessary.

3.1.2 Reservation webservice

Requests received through the Reservation-WS are processed by the *ReservationRequestHandler* class. This class serves as a facade towards the *eu.ist_phosphorus.nsp.reservation* package; its sole purpose is to distribute requests to the different handler classes responsible for specific tasks.

The *ReservationSetupHandler* processes requests that are related to the establishment of new reservations (cf. Section 2.1.2). This class checks the availability of network resources and sets up reservations. Both operations interact with the path computation module (cf. Section 3.4) and share some other functionality.

This is not the case for the other operations that are implemented in the *ReservationOperationsHandler*, which processes requests that are related to previously established connections. It retrieves reservation data, checks the status of reservations, cancels reservations and activates previously reserved services (in case the auto activation feature was not requested).

The MSS will use this interface to request the required operations related to reservations. The MSS must implement the Reservation-WS proxy in order to be able to invoke these operations, and thus, it must be also compliant with its XSD types. Hence, when the MSS needs for an operation of the WS, it has to create the WS request, fill it with all the required parameters and invoke the operation. The NSP will receive the request through the WS and if it is correct, will process it and send back the reply to the MSS. This reply will contain an XML object to be processed by the MSS. So, the MSS only has to know the location of the Reservation-WS and be compliant with its XSD types and services to be able to manage reservations in the NSP.

3.2 Request and exception handling

The webservice request, response and exception handling is abstracted in WP1 by using a unified request handling layer that was described in D1.4. This layer is used in the whole service plane. Its task is to create a transparent communication between the middleware (e.g. the Meta-Scheduling Service) and the service plane without dealing with webservice related issues. Further details are shown in Figure 3.1. Each webservice forwards the requests to its corresponding request handler. This handler cares about the data binding, exceptions and the request validation.

Using the above mentioned architecture simplifies the communication between the middleware and the service plane substantial and failures within the lower systems can be located in an efficient manner. Hence the request and exception handling is described again in this deliverable with a focus on the middleware-serviceplane-communication.

3.2.1 Data binding

The Java language is used in most middleware implementations. In order to abstract any XML related functionalities from the middleware, the Java Architecture for XML Binding (JAXB) [JAXB] is used. Incoming XML requests are transparently unmarshalled into annotated Java objects that can easily be used to retrieve or modify the submitted data in a well-known fashion. The Java response objects then are marshalled back into the according XML representation. This way, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure. The needed JAXB Java objects are generated automatically by using the XML Schema of the webservice request and response types. By using this construct also exceptions with useful debugging information can be exchanged transparent through the webservices.

3.2.2 Exception handling

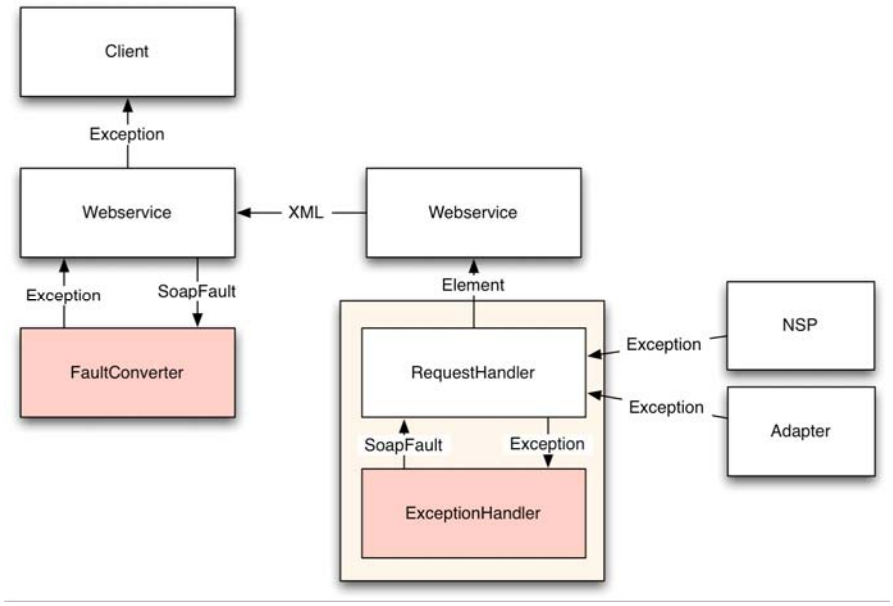

Figure 3.2: Exception handling

Exceptions are a construct of the Java programming language to handle expected or unexpected failures. A webservice has a similar approach and it uses so called SoapFaults to communicate failures.

As shown in Figure 3.2, if an exception is thrown by the service plane it is caught by the appropriate request handler and forwarded to the exception handler. The exception handler converts the Java exception including the message and the stacktrace to an XML stream. This stream, which contains the complete Java stackstrace in the SoapFault “Detail” field for debugging reasons, is sent out to the middleware (client) via the webservice in form of a predefined SoapFault (cf. Section 2). On the client side, the SoapFault is converted back to the original Java exception by the client’s webservice. This way the service plane can handle failures efficiently and communicate all the information to the middleware. From the client point of view, the called webservice operation acts as a normal Java method with exceptions.

3.2.3 Validation

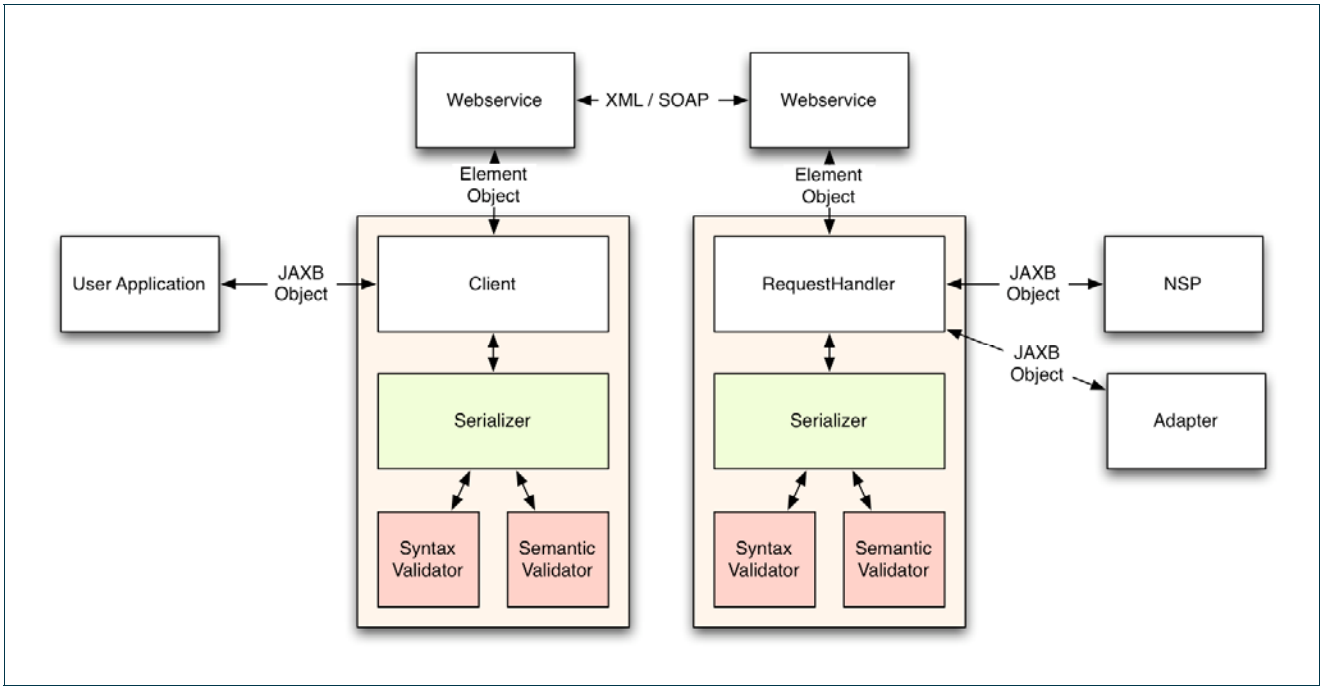

Figure 3.3: Request validation

Before sending or forwarding any (XML) data, the requests and responses are validated within the service plane. As shown in Figure 3.3, the incoming and outgoing messages are validated in two ways. First, the structure itself is validated by the syntax validator. The message must conform to the XML Schema definition of the corresponding type. If this is not the case, a syntax validation exception is thrown and the message is discarded. If the syntax is correct, the semantic validator verifies constraints regarding the content. If e.g. the requested minimum bandwidth is higher than the requested maximum bandwidth in a message, a semantic validation exception is thrown and again the message is discarded. This way, the core system can assume to retrieve only valid data. Also the middleware receives helpful information regarding the validity of the sent request immediately.

3.3 Authentication and authorization

As shown in Figure 3.4, the NSP contains a central module that is used to authenticate all incoming and to sign all outgoing traffic. This Message Level Security (MLS) service is based on the OASIS Webservices Security standard [WSS]. Besides procedures to sign and to encrypt SOAP messages the standard includes options to attach security credentials like username/password, X.509 certificates or tokens.

The MSS acts as an Attribute Authority (AA) that serves the role of a trusted entity for the service plane that mediates requests for holders of digital credentials. It must have privileged access to the local authentication domain database that holds information (identity attributes) about the credential holders. The MSS operates on rulesets defining what attributes can be attached to the request and under what circumstances.

The service plane itself authorizes the request based on the attached attributes and the signature. It forwards the request with the user credentials (attributes) that are contained in the incoming message from the middleware to the involved NRPS adapters and vice versa. The same is done within the adapters but here the service plane plays the role of the AA.

It is assumed that each domain has its own policy and attribute database. The NRPS adapter may map the global attributes to local ones or local user accounts. In the case that global and local attributes are identical, this mapping reduces to the identity function.

Since the communication with the service plane requires valid authentication credentials, the NSP creates a transitive trust relation between all participating partners. This is achieved by preinstalling the service plane public key in the MSS and in each NRPS adapter. On the other hand, the service plane must possess the public keys of each communication party in advance.

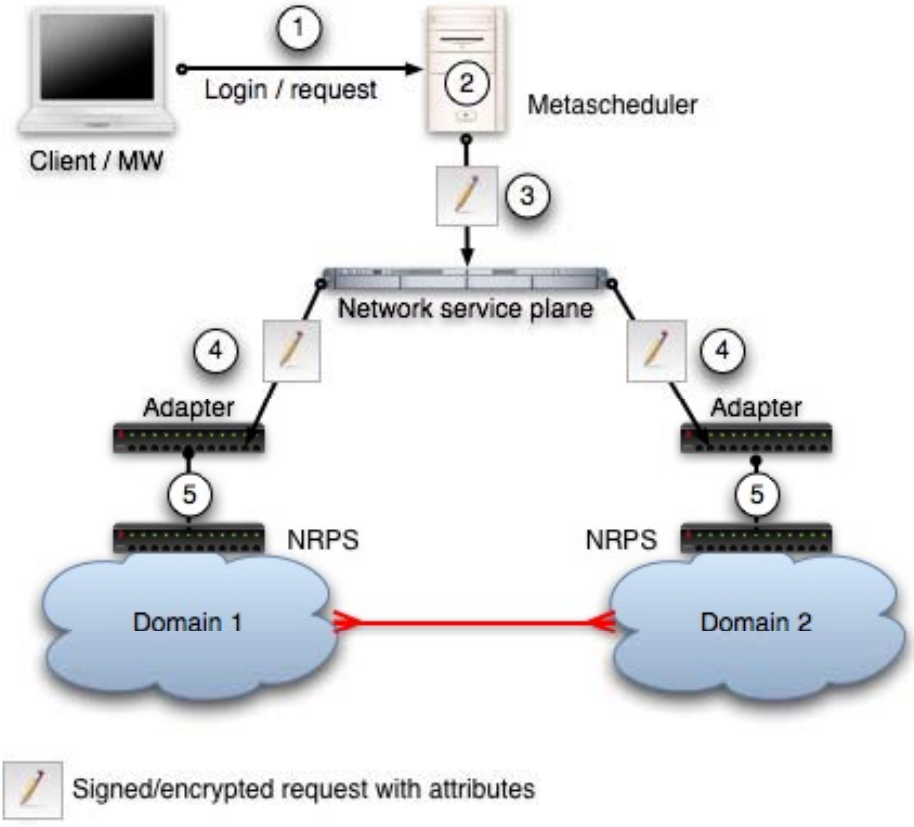

Figure 3.4: Authenticated message flow

Figure 3.4 depicts a sequence of interactions needed for the authentication flow mentioned above. The following sequence description is simplified and reduced to a single MSS-NRPS communication. The initial clients (or middleware's) request could also be sent to an NRPS adapter.

(1) An MSS client sends a request to the Meta-Scheduling Service (MSS) with local user credentials. (2) The MSS authorizes the user and the request locally. In case the request is authorized successfully, the scheduler maps the user credentials to accordant global attributes, adds these to the request to the NSP and signs the message with its private key. (3) The message then is sent to the NSP on behalf of the client. Since the public key of the MSS is trusted within the NSP, the message is accepted in the next step. The signature of the valid incoming request will be removed and the request may be split into several new requests. (4) The outgoing messages to the NRPS adapters are signed by the NSP. All authorization related information that may be added by the MSS is forwarded without any modification. In the expected case that the NRPS adapter trusts the NSP key all authorization information (e.g. global attributes, tickets) and the request are forwarded to the

specific NRPS. (5) A complex authorization process has to be implemented in the NRPS adapter or the NRPS itself.

This way, the service plane acts as a transparent broker between the MSS and each NRPS. It is self-evident that this message level security flow is also applied for the corresponding response messages. Additionally, this architecture could be used to encrypt the whole message flow.

3.4 Path computing

The Path Computer module of the NSP is designed to calculate interdomain paths using the Dijkstra algorithm. It can calculate multiple paths in one single path computation request. When calculating a path, blocking of resources (i.e. resources that are in use by another service that is at least partly overlapping in time) is taken into account. Each path is returned as a list of tuples of endpoints. Each tuple consists of two endpoints of the same domain. The calculation of the intradomain parts of a path is left to the Path Computer of the domains' NRPSs. This means that the MSS should specify network requests as source and destination endpoints. For this reason the only the source and destination endpoint need to be known to the MSS, but not the complete topology.

4 Network Service Plane functionality

4.1 Network resource availability query

The MSS can trigger an availability query by sending an *IsAvailable* request to the Reservation-WS.

Upon reception of an *IsAvailable* request, the corresponding routine in the *ReservationSetupHandler* class is called. Though a reservation will actually not be made, the task to be solved is very similar to an actual reservation, the only exception being that the queried resources are not reserved. Also, alternative start time offsets can be returned, while an actual reservation only either succeeds or fails.

The requested services are used as input for the *getAvailableServiceList* routine that is also used for *CreateReservation* requests. This routine queries the *PathComputer* for paths for all of the requested connections and splits the single multidomain request to multiple single domain requests, one for each of the involved domains. These requests are handed to the *NRPSManager* that takes care of sending these requests to the NRPSs and collecting the corresponding replies.

If the requested resources are not available in one or more of the involved domains, they are pruned from the *PathComputer* instance and the *PathComputer* is queried again for an alternative path. This process is repeated until either a suitable path is found, or until so many resources have been pruned that no path is available for one or more connections. In the first case, the requestor (MSS) is informed that the resources are available. In the latter case, the requestor (MSS) is informed that they are not available, and the earliest alternative start time offset of those recorded as described above is reported as an alternative start time offset.

4.2 Network resource reservation

The MSS can trigger a network resource reservation by sending a *CreateReservation* request to the Reservation-WS. The reservation of network resources is internally handled similar to the availability query described in the previous section. Before sending *CreateReservation* messages to the NRPSs, the availability of the requested resources is checked. This is to prevent a series of *CreateReservation* and *CancelReservation*

messages that would be necessary if one or more domains in a multidomain path are not able to fulfill the reservation.

Alternative start times reported by the NRPS adapters are however discarded. The availability of intradomain paths along different routes returned by the *PathComputer* is checked merely with the constraints specified in the *CreateReservation* message. In case a path consisting of domains that all gave a positive reply to the availability query is found, the final reservations for all intradomain paths are established.

4.3 Reservation status query

A *GetStatus* request is mapped to a set of single-domain *GetStatus* queries in a straightforward way. From the reservation ID that is part of this message, the domains and the reservation IDs used for this reservation inside each of the domains are retrieved from the database, and a set of corresponding *GetStatus* messages is constructed and passed to the *NRPSManager*.

Practical considerations during first testing of the code have lead to a slight modification of the *GetStatusResponse* messages. In addition to an overall status code for each connection that is generated from the set of status codes for this connection received from the participating NRPSs, the *GetStatusResponseType* optionally contains *DomainStatus* elements, each of which contains a domain name and an element of type *ConnectionStatusType*, i.e. the connection status received from the specified domain. This is mainly interesting for debugging purposes in cases where the status values are not consistent. E.g., in case a connection should be established, all domains should return the status code *active*. If one domain returns a different status code, it is immediately visible in which domain the error has occurred.

The MSS can invoke this operation in order to know the status of a previously created reservation. To do this, the MSS has to know the reservation ID and the Service ID to construct the *GetStatus* object that will be sent to Reservation WS of the NSP. The reservation ID and service ID used must be the ones that the MSS received in the *CreateReservationResponse* message from the NSP when the reservation was successfully created. This mechanism allows the MSS to retrieve detailed status information of a subset of services within a reservation. This way, the software on the MSS can monitor all the connections that it considers most critical for its purposes, as far as a service is defined as a set of connections.

4.4 Reservation cancellation / connection teardown

An already established reservation is cancelled by a *CancelReservation* message. For the NSP, it is not of importance whether the reservation contains services that are already active or whether all services are still waiting to be started. Once a *CancelReservation* is received from the MSS, all services on it are cancelled.

To cancel a reservation, the NSP looks up the intradomain reservations that were made for the input reservation and sends a *CancelReservation* message with the corresponding ID to each of the domains.

The MSS is able to cancel a reservation when needed by sending to the WS the *CancelReservation* request object with the identifier of the reservation to be cancelled. As in other operations, this identifier was stored in the MSS once the reservation creation process was successfully performed.

5 Conclusions

This deliverable describes integration of the NSP with the MSS. Transparently enabling multidomain advance reservation features, the NSP allows a network consisting of multiple administrative domains managed by different NRPSs to be integrated into the MSS.

To achieve this, the NSP offers a reservation webservice that is comparable to a NRPS webservice. An an adapter is used to mediate between the WS-Agreement protocol and messages and the interface of the NSP. Requests received across this interface are processed in such a way that the multidomain nature of the underlying network is hidden towards the MSS.

6 References

[Hibernate] Hibernate (Relational Persistence for Java and .NET), http://www.hibernate.org/

[JAXB] Java Architecture for XML Binding (JAXB) reference implementation, https://jaxb.dev.java.net/

[MSS] O.Wäldrich, Ph. Wieder, and W. Ziegler, A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resources. In Proc. of the Second Grid Resource Management Workshop (GRMWS'05) in conjunction with Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Lecture Notes in Computer Science, Volume 3911, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski (eds.), pp. 782-791, Springer, Poznan, PL, September 11-14, 2005. ISBN: 3-540-34141-2.

[WSN] S. Graham, D. Hull, B. Murray (editors), "Webservice Base Notification 1.3 (WS-BaseNotification)," OASIS Standard, October 2006

[WSS] A. Nadalin, C. Kaler, P. Hallam-Baker, R. Monzill et al., "Webservices Security: SOAP Message Security 1.0 (WS-Security 2004)," OASIS Standard, vol. 200401, 2004.

7 Acronyms

AAA	Authorization, Authentication and Accounting
ARGON	Allocation and Reservation in Grid-enabled Optic Networks
BoD	Bandwidth on Demand
CP	Control Plane
DB	Data Base
DRAC	Dynamic Resource Allocation Controller
E2E	End-to-End
GLIF	Global Lambda Integrated Facility
GMPLS	Generalized Multi Protocol Label Switching
IDM	InterDomain Manager
ID	Identifier
IP	Internet Protocol
I-NNI	Interior NNI
LSP	Label Switched Path
MSS	Meta-Scheduling System
NNI	Network-Network Interface
NRPS	Network Resource Provisioning System
NSP	NSP
NSAP	Network Service Access Point
OSPF	Open Shortest Path First
QoS	Quality of Service
TNA	Transport Network Address
UCLPv2	User Controlled LightPaths version 2
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
WP	Work Package
WS	Webservice