034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.4.1

AAA Architectures for multi-domain optical networking scenario's

Due date of deliverable: 30-09-2007
Actual submission date: 01-10-2007
Document code: <Phosphorus-WP4-D.4.1>

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
University of 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 result of the development of the AAA Authorisation infrastructure for multidomain Optical Network Resources Provisioning (ONRP). The proposed architecture attempts to address key access control problems when integrating heterogeneous Network Resource Provisioning Systems (NRPS) being deployed in the different Phosphorus testbeds. The proposed architecture also targets to ensure future compatibility with the Grid and NREN access control solutions and infrastructures.

This deliverable relies on the WP4 M4.1 milestone report that provides general and technical information about available concepts, standards and technologies in network and application access control with the special attention how these technologies can be used for on-demand network reservation and managing dynamic security services.

The report summarises recent research and developments of the Generic AAA Authorisation framework (GAAA-AuthZ) to support Complex Resource Provisioning (CRP) and in particular focuses on the support of the major Phosphorus use cases. The document provides an overview of the WP1 NSP AAA/AuthZ issues and discusses how the GAAA-AuthZ can be implemented in the G.OUNI/G2MPLS (being developed in the WP2) to address GLIF UNI1.0 recommendations.

The report provides general design recommendations and suggestions to the GAAA-AuthZ for ONRP which are used to compile an extensive list of detailed requirements to different components of the developed GAAA-AuthZ infrastructure and its initial implementation in the framework of the GAAA Toolkit.

The proposed GAAA-AuthZ architecture identifies key functionalities to support multidomain ONRP and introduces a number of mechanisms and solutions to support them. The proposed architecture allows smooth integration with other AuthZ frameworks as currently used and developed by NREN and Grid communities.

The report provides detailed technical information about current development and implementation of the Token Based Networking (TBN) and ForCES architecture which is being developed in cooperation between the University of Amsterdam and the University of Patras as a part of the WP4 activity.

It is intended that this report will provide a technological and technical basis for adding AAA/AuthZ services to ONRP technologies and components being developed by other Phosphorus packages. Proposed GAAA-AuthZ solutions and current implementation in GAAA Toolkit can be used in basic network provisioning scenarios and frameworks such as NRPS and Grid GMPLS.

Table of Contents

0 Executive Summary

The Authentication, Authorisation and Accounting (AAA) service is an important component of the supporting infrastructure for on-demand Optical Network Resource Provisioning (ONRP) across multiple domains and different target consumer applications. A consistent AAA infrastructure requires interactions of the related AAA components at all networking layers including the network/forwarding elements, the control plane, the reservation and provisioning service, and the user/target application layer.

This deliverable describes the result of the development of the AAA Authorisation infrastructure for multidomain Optical Network Resources Provisioning (ONRP). The proposed architecture attempts to address key access control problems when integrating heterogeneous Network Resource Provisioning Systems (NRPS) being deployed in the different Phosphorus testbeds. The proposed architecture also targets to ensure future compatibility with the Grid and NREN access control solutions and infrastructures.

This deliverable relies on the WP4 M4.1 milestone report that provides general and technical information about available concepts, standards and technologies in network and application access control with the special attention how these technologies can be used for on-demand network reservation and managing dynamic security services.

The report summarises recent research and developments of the Generic AAA Authorisation framework (GAAA-AuthZ) to support Complex Resource Provisioning (CRP) and in particular focused on the support of the major Phosphorus use cases. The document contains an overview of the WP1 NSP AAA/AuthZ issues and provides an analysis how GAAA-AuthZ can be implemented in the G.OUNI/G2MPLS (being developed in the WP2) to address GLIF UNI1.0 recommendations.

The report provides general design recommendations and suggestions to GAAA-AuthZ for ONRP which are used to compile an extensive list of detailed requirements to different components of the developed GAAA-AuthZ infrastructure and its initial implementation in the framework of the GAAA Toolkit.

The proposed GAAA-AuthZ architecture identifies key functionalities to support multidomain ONRP and introduces a number of mechanisms and solutions to support them. The proposed architecture allows smooth integration with other AuthZ frameworks as currently used and developed by NREN and Grid communities.

The report provides detailed technical information about current development and implementation of the Token Based Networking (TBN) and ForCES architecture which is being developed in cooperation between the University of Amsterdam and the University of Patras as a part of the WP4 activity.

It is intended that this report will provide a technological and technical basis for adding AAA/AuthZ services to ONRP architectures and components being developed by other Phosphorus packages. Proposed GAAA-AuthZ solutions and current implementation in GAAA Toolkit can be used in basic network provisioning scenarios and frameworks such as NRPS and Grid GMPLS.

1 Introduction

The main objective of the Phosphorus project is to address some of the key technical challenges to enable on-demand e2e network services across multiple administrative and security domains. The Authentication, Authorisation and Accounting (AAA) service(s) is considered as an important component of the supporting infrastructure for on-demand Optical Network Resource Provisioning (ONRP) across multiple domains and different target consumer applications. A consistent AAA infrastructure requires interaction of the related AAA components at all networking layers including network/forwarding elements, control plane, reservation and provisioning service, and user/target applications layer.

This deliverable describes the result of the development of the AAA Authorisation infrastructure for multidomain Optical Network Resources Provisioning (ONRP). The proposed architecture attempts to address key access control problems when integrating heterogeneous Network Resource Provisioning Systems (NRPS) being deployed in the different Phosphorus testbeds. The proposed architecture also targets to ensure future compatibility with the Grid and NREN access control solutions and infrastructures.

The report is organised as follows. Section 2 summarises recent research and developments of the Generic AAA Authorisation framework (GAAA-AuthZ) and provides general design recommendations and suggestions for GAAA-AuthZ for ONRP. This section refers to the WP4 M4.1 milestone report that provided extended analysis and technical information about available concepts, standards and technologies in network and application access control with the special attention how these technologies can be used for on-demand network reservation and managing dynamic security services.

Section 3 provides a general overview of the WP1 Network Service Plane (NSP) AAA/AuthZ issues and discusses how GAAA-AuthZ can be implemented in the G.OUNI/G2MPLS (being developed in the WP2) to address GLIF UNI1.0 recommendations. The section also provides a short overview of the eduGAIN framework being developed in cooperation between European National Research and Education Networks (NREN) and the GEANT2 project. The eduGAIN is considered as a recommended common Authentication and Authorisation Infrastructure (AAI) for interdomain network resource provisioning between NREN's. Section 3 concludes with the extensive list of detailed technical requirements that should be taken into account and considered when developing GAAA-AuthZ infrastructure for multidomain ONRP.

Section 4 describes the proposed GAAA-AuthZ architecture for general Complex Resource Provisioning (CRP) and ONRP in particular. It identifies key functionalities to support multidomain ONRP and introduces a number of mechanisms and solutions to support them, in particular: AuthZ ticket format for extended AuthZ session management, Token Validation Service (TVS) to enable token based policy enforcement, policy Obligation Handling Reference Model (OHRM), and XACML policy profile for OLPP. The proposed architecture will allow smooth integration with other AuthZ frameworks as currently used and developed by NREN and Grid community.

Section 5 describes how the proposed GAAA-AuthZ architecture is implemented in the current version of the GAAA Toolkit. It provides general description of the GAAA Toolkit structure and functionalities to support network resource provisioning and more detailed description of such components as TVS and GAAAPI that can be used as a pluggable component to add AAA/AuthZ services to different NRPS frameworks.

It is intended that the proposed GAAA-AuthZ architecture and its implementation in GAAAPI and TVS component will provide a technological and technical basis for adding AAA/AuthZ services to ONRP architectures and components being developed by other Phosphorus packages.

Section 6 provides detailed technical information about current development and implementation of the Token Based Networking (TBN) and ForCES architecture which is being developed in cooperation between the University of Amsterdam and the University of Patras as a part of the WP4 activity.

Finally, section 7 provides summary of the current results and suggests further developments.

2 Generic AAA Authorisation Framework (GAAA-AuthZ)

This section describes the major concepts of the Generic Authentication, Authorization and Accounting (GAAA) Authorisation Framework [1, 2] used to describe more complex authorization sequences enabling the access and usage of dynamically provisioned network resources. The section provides general GAAA-AuthZ implementation suggestions and specific suggestions for on-demand Optical Network Resource Provisioning.

2.1 Authentication, Authorization, and Accounting (AAA)

Authentication, authorization, and accounting (AAA) is a term used to refer to a framework for intelligently controlling access to resources, enforcing policies, auditing usage, and providing the information necessary to bill for services. These combined functions are considered important for effective network management and security.

Authentication (AuthN) and Authorisation (AuthZ) are the components of the access control function to ensure that access to the resource or service is granted to the access subject (human, service or process) that has right to use the resource and perform those operation on the resource that it is allowed. Initially Authorisation and Authentication frameworks were defined in the ITU-T standards X.812 and X.811 [3, 4].

Authentication is the process of identifying a user or an access subject, based on identity credentials, e.g. username and password, digital certificates, or one-time-tokens. Authentication refers to the confirmation that a user/subject who is requesting services is a valid user of the resources or services requested. Typically AuthN involves comparing a user's authentication credentials with the user credentials stored in a database or the AuthN/AAA service, or checking validity of the user credentials obtained from the trusted AuthN service or trusted Identity Provider.

Based on positive AuthN, a user must obtain authorization for doing certain tasks. Authorization is the process of granting or denying a user access to resources once the user has been authenticated. The amount of information and the amount of services the user will be granted depends on the user's authorization level which is defined by the user attribute credentials. In other words, Authorization is the process of enforcing policies: determining what types or qualities of activities, resources, or services a user is permitted. Usually, authorization occurs within the context of authentication. An authenticated user is provided with the attributes that are required for the authorisation decision.

Accounting is the process of keeping track of a user's activity while accessing the resources or services. Accounting is carried out by logging of session statistics and usage information. This data can be used for trend analysis, capacity planning, billing, auditing and cost allocation.

2.2 Basic GAAA Authorisation framework operational models

The Generic AAA Authorisation Framework and its specific implementations for network provisioning define three basic operational models that describe interaction (in sense of request/response sequences) between a user, a service or resource provider and AAA Authorisation service acting as an Authority [1, 2]. These sequences have also been used as basis for the Conceptual Grid Authorization Framework and Classification document [5].

The push authorization sequence.	Within the push (or token-) sequence, the User first requests an authorization from a trusted Authorisation service that may or may not honour the User's request. It then may issue and return some kind of Authorisation assertion (a secured ticket or token) that acts as a proof of right or as asserted list of requestor capabilities. Typically such an assertion has an associated validity time window. The assertion may subsequently be used by the User to request a specific service by contacting the resource. The resource will accept or reject the authorization assertion and will report this back to the requesting Subject. The resource must have been provisioned with the appropriate key material to recognize the appropriate assertions.
The pull authorization sequence.	Within the pull (or outsource-) sequence, the User will contact the resource with a request. Before admitting the service request, the resource must contact its Authorization service. The Authorization service will evaluate the request against a specific authorization policy and will return an authorization decision. The resource will subsequently grant or deny the service to the User by returning a result message. The resource, which enforces a policy, effectively out-sources a policy decision.
The agent authorization sequence.	Using the agent (or provision-) sequence, the User will contact an Agent, which will handle the User's request for the particular resource. The Agent is trusted both by the User and the resource. The Agent will make an authorization decision and, using its own or User-delegated credentials, it will contact the resource to provision the requested service. The Agent will provide the User with details on how to contact and use the Service.

The three basic authorisation sequences described above are elementary abstractions of more complex real world examples that normally combine the basic sequences. It may use various protocols and message formats to handle and secure user credentials and requests.

Although more functions can be found in both an Authority and a resource, an authority typically acts as a Policy Decision Point (PDP), and a resource incorporates a Policy Enforcement Point (PEP) which enforces the policy decision it has obtained from the Authority. In the subsequent discussion we may use the term PDP and PEP to represent functions inside the corresponding entities.

Some examples of combining basic authorisation models to achieve performance or security benefits are discussed below in relation to two major GAAA implementations for on-demand Complex Resource Provisioning (CRP) [6, 7] and for policy-based access control in collaborative applications [8, 9].

2.3 GAAA-AuthZ operational models for complex resources

Two basic use cases/models are discussed in this section:

1) the combined agent-push (provisioning) model for complex resources, and

2) the combined pull-push model for multi-layer resource protection.

The current research is aimed at the development of operational models based on the GAAA tools to provide access and usage control of a complex set of resources in a distributed heterogeneous environment. Such an environment can be characterized by the following:

- Access control and usage policies are defined by multiple policy instances, governed by different authorities and stored in different formats. Such environment can however be structured and ordered as a combined policy.

- Multiple PDPs and PEPs may interact in sequences, which can either be flexibly configured or pre-defined. The sequences can be described using elements of the GAAA authorization framework.

- A network of PDPs and PEPs can operate in the push-, pull and agent modes. Many access control models may require the agent mode to be supported by push functionalities. in a particular case of the general resource provisioning the first stage of the resource discovery and reservation typically uses the agent mode while the actual resource or service access is supported by pushing reservation and/or authorisation credentials, e.g. in a form of authorisation ticket/token.

- PDP and PEP elements can be part of a resource, user or a service. A set of PEPs and PDPs can together create a distributed Access control infrastructure.

An important component of both combined models is the use of authorisation tickets and tokens for security context handling and performance optimisation. Authorisation tickets and tokens also provide a mechanism for authorisation session management.

Figure 2.1 illustrates an abstract access control model that combines two generic AAA Authorisation sequences: The agent sequence and the push sequence. Such a model is typically found within Bandwidth-on-Demand (BoD) or Optical Light Path Provisioning (OLPP) use cases that typically include two stages: The reservation stage, at which a reservation ticket in a form of AuthZ ticket or token can be created, and the access stage, at which the user can access the reserved resource by presenting the reservation ticket or token. The type of complex service that is provisioned is less relevant and this model can therefore be applied more generally.

In the agent model, the PDP orchestrates a (complex) service request on behalf of the Requestor. The policy, in such case, can be considered as a "driving policy" and as such represents elements of the total workflow of the system [6]. In case of a complex resource/service request, a sequence of PDPs may create a flow of recursive policy evaluation chains. The PDPs may use a set of PEPs to enforce the policy at different resources and services. It is assumed that each PDP can request other PDPs for evaluating some of the whole request components for the specific resource. PDP and PEP interaction is discussed below in more detail for the combined pull-push model.

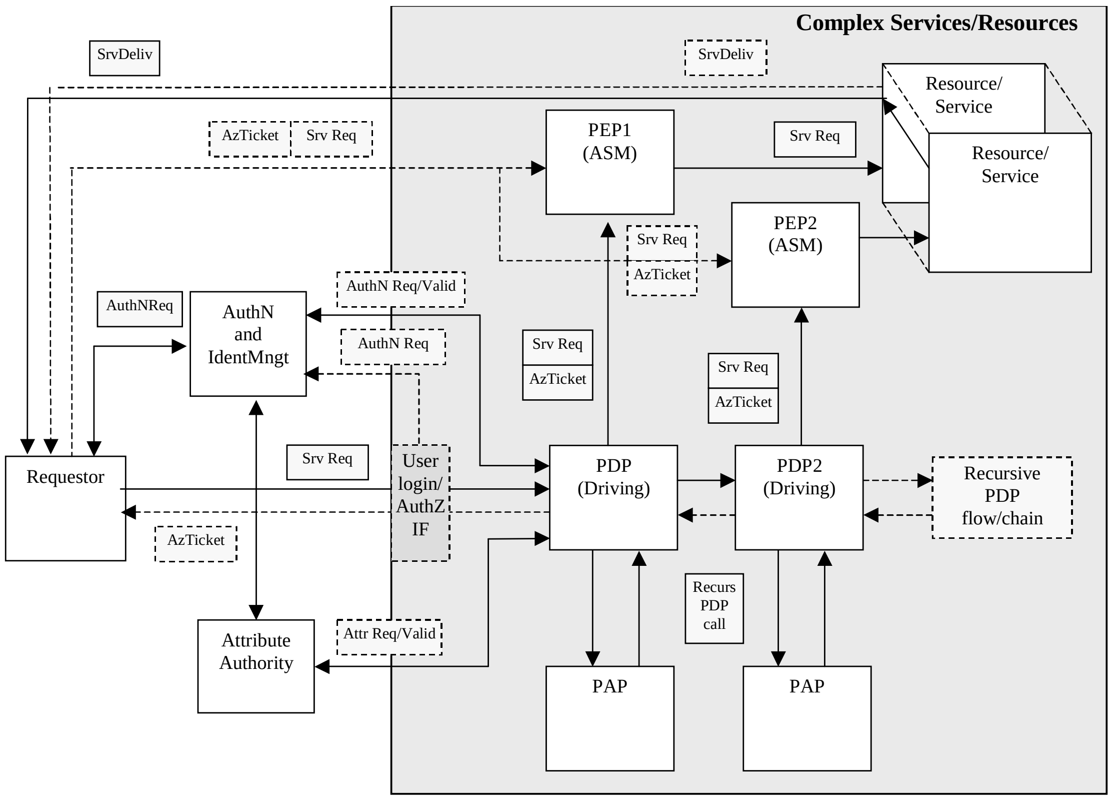

Figure 2.1. Major components of the complex resource/service authorisation service (combined push and agent model, complex/multi-component resource)

Figure 2.2 illustrates a typical policy based authorisation model that implements the pull model of the generic AAA Authorisation framework and may also use the authorisation ticket “push” functionality to optimise performance. The picture also explains how the policy combination can be done via PEP chaining/sequencing and/or PDP nesting/recursion as a common component for all GAAA operational models.

A detailed policy enforcement process analysis must formulate security constraints for a use case that involves evaluation of multiple policies with a combination of multiple PEPs and PDPs. The aim of such analyses is to preserve a site or resource access control integrity.

The proposed approach retains integrity of the combined policy based decision. Although the PDP, when evaluating a request from the PEP, may call for external evaluation of some other policy components, it will make its own final decision and will return a reply to the calling PEP only, which acts as a gateway to the initial request.

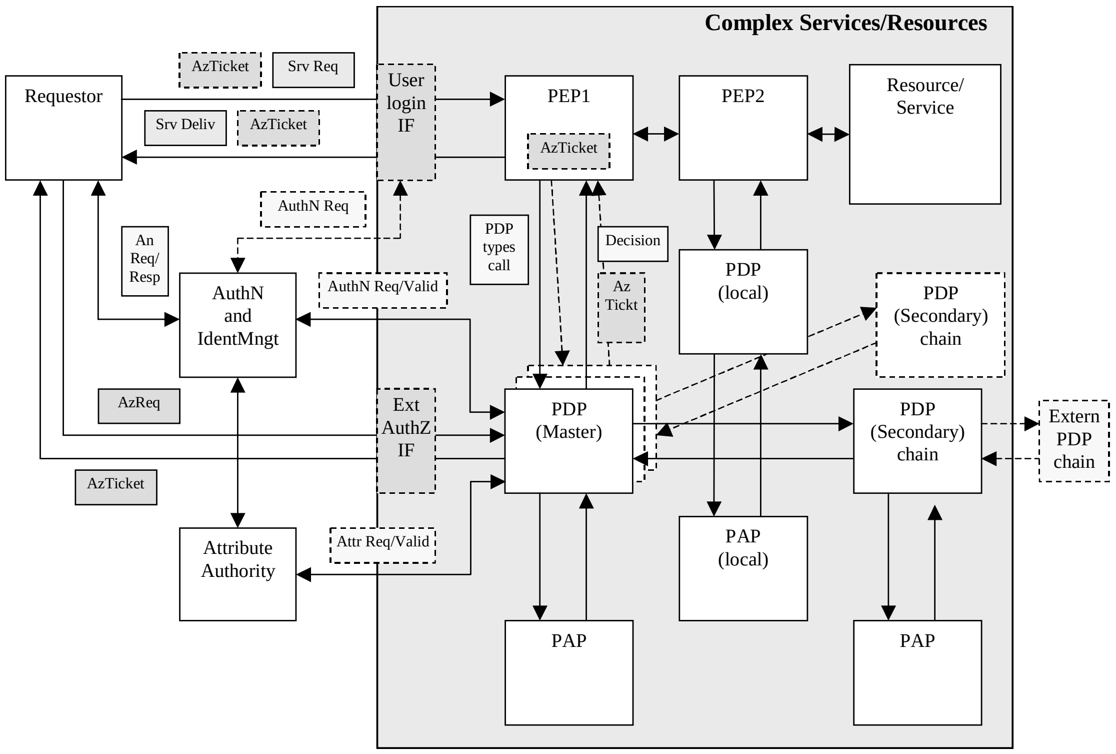

Figure 2.2. Multiple/multi-domain policies combination in complex resource/service Authorisation service (combined pull-push model)

The Requestor requests a service by sending a service request ServReq to the resource’s PEP providing as much or as little information about subject/requestor, resource, action, and additionally environment as it decides necessary according to used authorisation model and (known to the Requestor) local policies.

In a simple scenario, the PEP sends the decision request to the (designated) PDP, and after receiving a permissive reply from PDP, it relays a service request to the resource. The PDP identifies the applicable policy instance, retrieves required context information and evaluates the request against the policy. During this process, it may need to validate the presented credentials locally based on pre-established/shared trust relations, or call external Authentication and Attribute Authorities.

The process described above represents a basic scenario. However, in a more complex and open environment, the PEP may receive requests that have different formats and semantics (namespaces) or may refer to policies stored in other policy repositories. In such case, the PEP should have a possibility to relay a decision request to an appropriate PDP capable of handling the entire decision request. It is essential that a request is evaluated as a whole and an ultimate decision is made by a single PDP. This PDP may however make calls to external PDPs to evaluate some request components and process their decisions as components of the general policy evaluation process. The PDP that makes a final combined decision can be defined as a master PDP and it needs to have mechanisms in place to preserve integrity of the final combined decision. In case an integral request evaluation is not possible, responses such as “not applicable” with the detailed problem reporting should be possible.

Existing (open) policy expression formats, such as XACML and our AAA driving policy language provides mechanisms that allow a particular policy instance to refer to another policy instance. Complex combined policies can be created by a Policy Authority Point (PAP) on a PDP policy request, or processed by the PDP by requesting required policy components during the request evaluation.

As a trade-off of being open by using separate access control components and open standards, the solution above has known performance concerns. The resolution of this problem is seen in combining pull and push operation models. Since the decision is made by the PDP, an AuthZ ticket can be issued and used in the next similar or repetitive actions requests for the duration of a ticket’s validity period. An AuthZ ticket can be obtained from the PEP during the first access request or it can be requested from the PDP via an external AuthZ interface prior to sending a service request.

In the push model, the Requestor first requests an Authorisation decision to obtain an AuthZ ticket which it will attach to one or more subsequent service requests. The PEP will evaluate the authenticity, integrity and validity of the presented ticket and maybe some additional security credentials that prove correctness of e.g. the ownership, billing information, or service level. However, no other access decision functions should be given to the PEP as a functional component. If there is a need to enforce other components of the site or resource control, like a “blacklist”, it should be done via separate (local) PEP-PDP chain.

2.4 General GAAA-AuthZ implementation suggestions

Section 2.3 describes rather simple scenarios, but they already require that both requestor and resource services know explicitly or implicitly the policy, semantics and know or can access the context information. The Requestor and the resource should have established a trust relation via common PKI or preliminary shared public and secret keys.

When implementing an authorization sequence, the following issues should be considered (using RFC2119 terminology for the words MUST, SHOULD, MAY etc.):

(1) A PEP typically runs as a part of the resource service/process and therefore belongs to the resource trust domain. A PDP MAY run as a part of the resource service/process and therefore be a part of the resource trust domain, or it MAY be called by the PEP and have pre-established trust relations with the PEP/resource.

(2) PDP and PAP MUST share a common namespace.

(3) A policy and additionally a PAP SHOULD be referenced in the request message explicitly, known to PEP and PDP a priori (pre-configured), or derived from the request context.

(4) A resource SHOULD have a possibility to request different PDP types based on the type of service request. This can be done by using a chain of specialised PEPs connected to specialised PDPs, or by providing a PEP with the ability to request different PDPs depending on the request context (e.g., semantics/namespace, referred policy).

- By definition, a PEP MUST have the ability to recognise a request's context (semantics/namespace) and convert the initial request format/semantics to those accepted by a particular PDP that will handle a particular request.

(5) Different parts of the whole AuthZ request MAY be evaluated either by relevant/applicable PEP-PDP pairs or by multiple specialised PDPs. The following combination rule must be applied to ensure evaluation process integrity and final decision consistency:

- In the first case, individual PEPs SHOULD constitute a chain that conjuncts individual PDPs decisions and MAY combine or aggregate other decision's components such as obligations.

- In the second case, a PEP SHOULD call the single master PDP that will call individual PDPs and combine a common decision according to a predefined combination algorithm.

- Only one (or the master) PDP MUST provide a final decision on the whole request received from a PEP.

(6) In general, AuthZ tickets and/or tokens can be used to support AuthZ session management functionality, and in particular to support complex pull and push AuthZ models for performance optimisation as a part of the general AuthZ session management.

- An AuthZ ticket typically is issued by the PDP based on a positive policy decision and relayed back to the Requestor. The Requestor will store the ticket/token and use in the following service requests.

- An AuthZ ticket MAY be issued by the PEP based on the fact that the PEP belongs to the resource trust domain and the ticket is actually a confirmation of the decision which the resource SHOULD/will trust. For future validation of the AuthZ tickets, the PEP MAY cache tickets locally to speed-up the validation procedure.

- An AuthZ ticket MUST contain or reference all information about the decision and the resource and SHOULD have validity and usage restrictions. Depending on the used security context management model, the AuthZ ticket MAY also include all context information about Requestor, its capability/attributes, and its Identity credentials (in a form of AuthN or Identity provider token).

- When using AuthZ tokens which uniquely reference AuthZ tickets but are smaller and simpler, AuthZ tickets SHOULD be cached by a PEP for future token resolution (or retrieval by token reference).

(7) AuthZ session management and AuthZ ticket/token handling functionality is subject to mutual agreement between the resource, PEP, PDP and the Requestor.

- A PEP SHOULD understand and have a possibility to validate an AuthZ ticket issued by a trusted PDP or AuthZ system in general.

- A Requestor SHOULD have the possibility to store or cache AuthZ tickets/tokens “as-is” and preserve their format and content unchanged (this is especially important for XML based tickets and tokens) and have the possibility to attach them to the related request type. Cookie handling functionality in web browsers is an example of such functionality.

(8) In the particular case of a dynamic access control policy operational model (so-called “push-policy”), an AuthZ ticket MAY be provided in the form of a (serialised) policy instance that defines exact matching conditions for the request evaluation. In this case, request processing SHOULD require only simple operations that can be executed by a PEP with the extended functionality for AuthZ ticket handling..

(9) A PDP is considered to be stateless to comply with most existing policy enforcement models and to remain compatible with existing PDP implementations. Stateful policy decision should be provided by special external handlers that enforce conditional policy decisions by executing returned conditions.

- In the XACML policy model, conditional policy decisions and stateful policy enforcement models are supported with so-called “Obligations” defined as actions that must be taken by the PEP following a positive or negative policy decision. Obligations are a part of the XACML policy and returned by the PDP “as-is”. It is the task of obligation handlers to process Obligations.

- In the generic AAA-AuthZ operational model, Obligations can be a part of the general AuthZ session management and included into an AuthZ ticket.

2.5 Specific GAAA-AuthZ implementation suggestions for on-demand Optical Network Resource Provisioning

Because the NRPS operation and OLPP provisioning process includes at least three stages (lookup, reservation and service access/delivery), the following specific issues should be considered:

- User/requestor credentials and consequently the trust model MAY be different at the reservation stage and at the provisioning stage.

- A reservation ticket for the complex network resource, used at the resource access/consumption stage, MUST include or reference all individual reservation tickets for the whole reserved resource.

- Multidomain OLPP requires inter-domain trust management that SHOULD be solved by establishing a general/common security federation or managed via delegation between inter-operating domains.

- Interdomain trust management MAY be implemented by using an open trust introduction model, for example DNSSEC or a Trusted Computing Platform infrastructure.

- To provide fast access to the reserved network resources, GAAA-AuthZ architecture SHOULD support both control plane and in-band policy enforcement mechanisms that can use AuthZ tickets or tokens (binary or XML based tokens).

    - An in-band policy enforcement mechanism can be achieved with the Token-based Networking Architecture developed as a part of the WP4 package.

    - To ensure the overall architecture consistency, both token-based enforcement models/mechanisms should use the same API.

- Given a multistage network resource provisioning process, the Obligations concept together with AuthZ ticket and token based policy enforcement mechanisms allow for flexible access control policy management but in turn will require multistage Obligations handling process to bridge between the stateless policy model and delayed usage of the reserved resource.

- The typical multi-user character of using network resources will require flexible user groups and hierarchy management, in particular to be able to support permissions and policy management delegation.

- The multi-domain character of network resource provisioning may require the combination and evaluation of multiple policies and the handling of multiple credentials that may differ by format, semantics and issuing authority.

Section 3.4 provides detailed technical requirements to the AAA/AuthZ services for Optical Network Resource Provisioning.

3 Overview of the Optical Network Resource Provisioning Use Cases and AAA/AuthZ Requirements

This section provides a short overview of other Phosphorus developments and documents to understand general and specific requirements to the AAA/AuthZ infrastructure for multidomain ONRP. Also, a short overview of the GEANT eduGAIN infrastructure and service is provided to understand how the Phosphorus AAA/AuthZ infrastructure can be integrated with other networks Authentication and Authorisation Infrastructure (AAI) using services provided by eduGAIN. The section concludes with the general requirements to the AAA/AuthZ infrastructure and services.

3.1 Access Control and Security in the WP1 Network Service Plane

In WP1, interfaces for the integration of NRPSs into the test-bed of WP6 are defined. Additionally, a Network Service Plane (NSP) for network resource interoperability is developed. This section gives an overview of the proposed AAI from the WP1 point of view.

As shown in Figure 3.1, 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 Web Services Security standard [10]. 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 NSP itself does not provide complex authorization or accounting mechanisms. Rather, it forwards attributes that are contained in the incoming message from the middleware to the involved NRPS adapters and vice versa. The authorization process is in the scope of each NRPS and the middleware – authorization tickets are transparently communicated through the NSP. 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 case global and local attributes are identical, this mapping reduces to the identity function.

Since the communication with the NSP requires valid authentication credentials, the NSP creates a transitive trust relation between all participating partners. This could be achieved by pre-installing the NSP public key in the middleware and in each NRPS adapter (similar to eduGAIN Bridging Elements described in section 3.3). On the other hand, the NSP has to know the public keys of each communication party in advance.

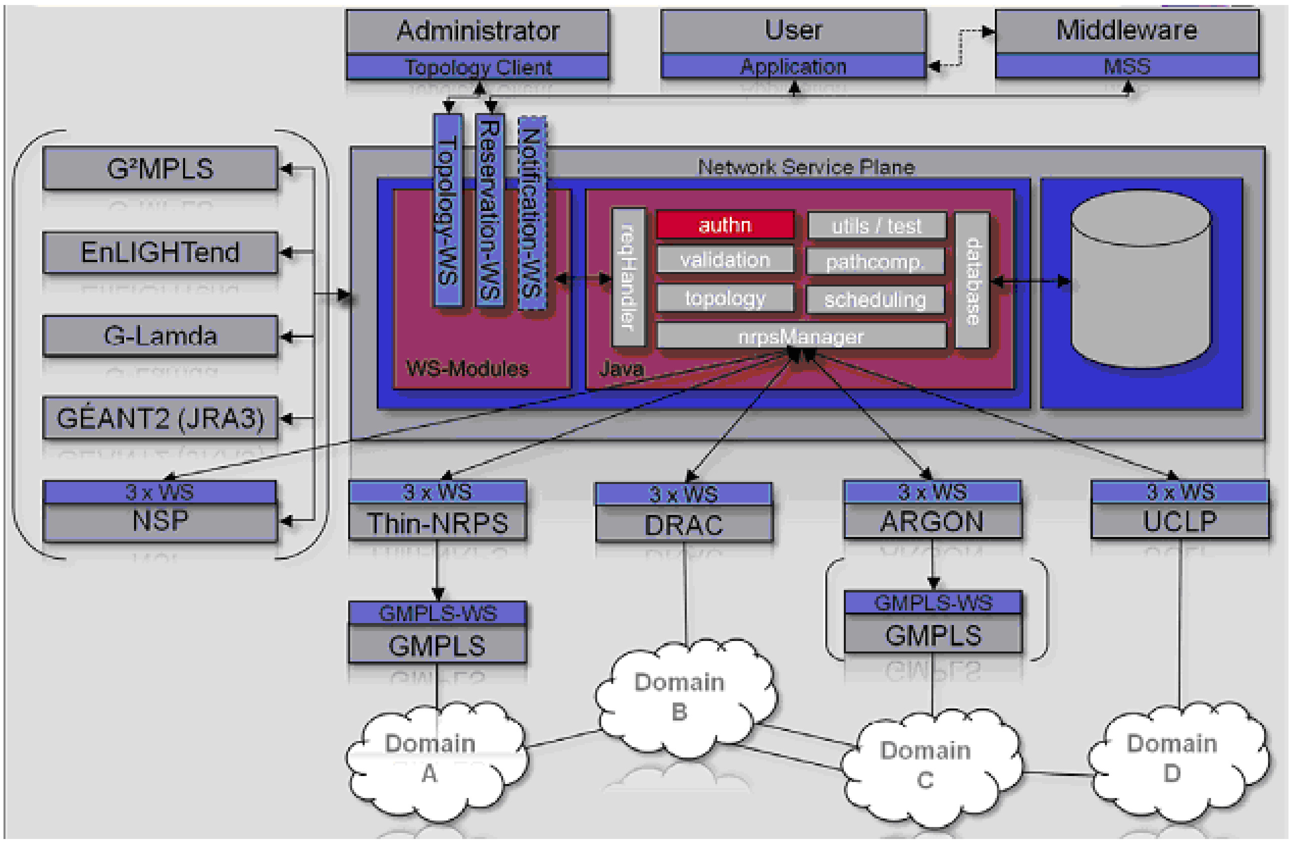

Figure 3.1. Proposed WP1 architecture with an authentication module

Figures 3.2 and 3.3 give an overview of the authenticated message flow mentioned above. The following sequence description is simplified and reduced to a single MSS-NRPS communication. The initial client request could also be sent to an NRPS.

(1) A 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 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)(6) 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)(7) A complex authorization process has to be implemented in the NRPS adapter or the NRPS itself.

It was shown that the NSP acts as a transparent broker between the middleware and each NRPS. 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.

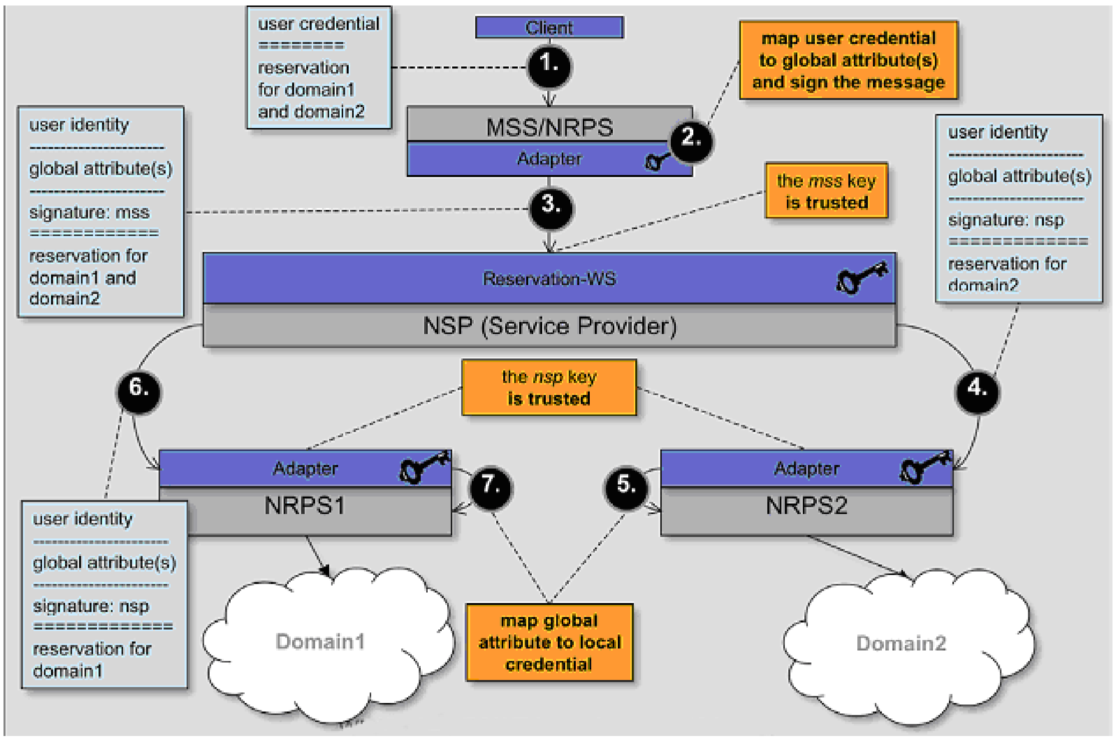

Figure 3.2. WP1 authentication workflow overview

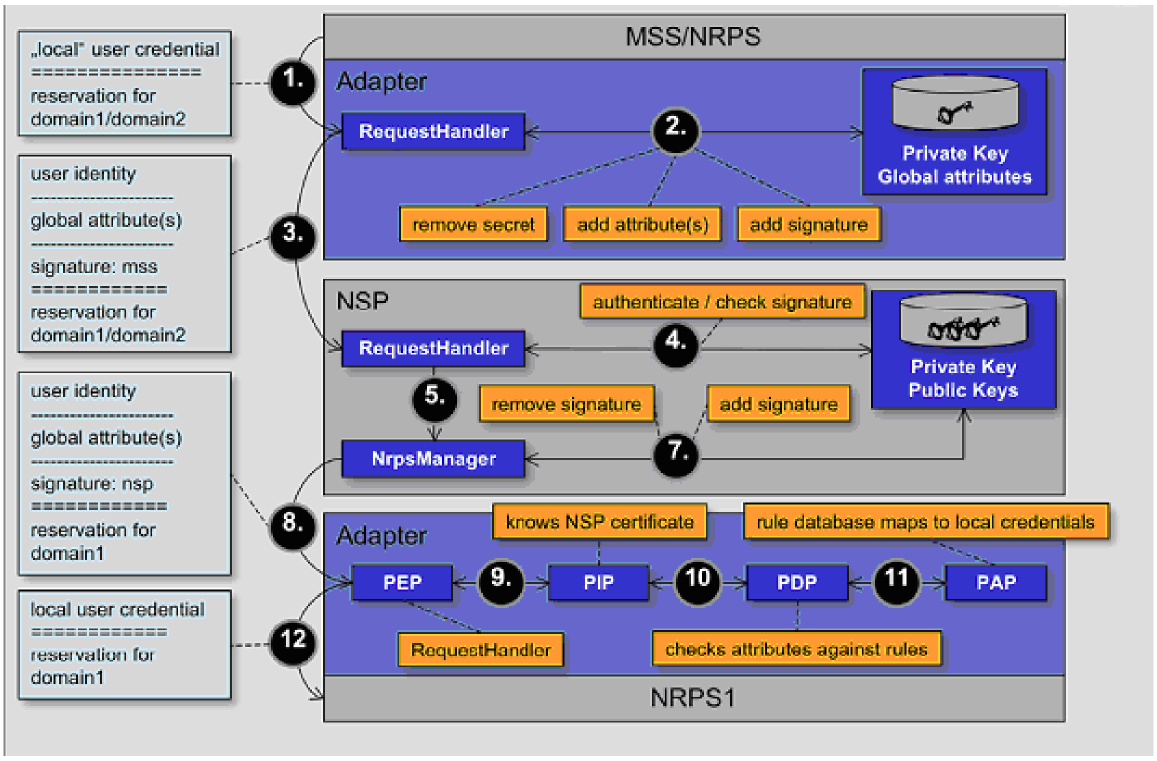

Figure 3.3. WP1 detailed authentication workflow

3.2 Policy Based Access Control in G.OUNI/G2MPLS

OIF User Network Interface (UNI) 1.0 Signalling Specification (OIF_UNI-01.0) [11, 12] provides UNI policy and security considerations. The UNI specification requires that the transport network must provide appropriate mechanisms to ensure accurate and authorised usage of network resources and client accountability. Access control and accounting should be governed according to the policy. A policy typically contains a collection of rules and/or conditions according to which the resources can be accessed or used. It is also suggested that policy based access control and usage should provide necessary information for accounting network related user activity.

As described in the UNI specification, policy rules should define conditions on parameters such as source and destination address, priorities, bilateral agreements between service providers, time or cost constrains, etc., which are all transport network related. A provided example suggests using user groups as user related attributes in a form like this: “Approve all requests on behalf of a given user group received from a given UNI-C agent, if the identity of the requestor can be verified”.

However, for Grid based collaborative applications in research and high-tech industry, network access is typically bound to the project based user and resource association such as a Virtual Organisation (VO) or a Virtual Laboratory (VL) and depends on user attributes. User attributes are typically maintained by a user Home Organisation (HO) or a VO attribute service such as VOMS [19]. It is also important that network resources are provided depending on the type of application that will use them. When integrated with the Grid resources or services access control, there is a need to make Grid and network resource access control policies compliant or compliant and compatible and possibly use the same access control tools and policy format.

The UNI specification suggests to use a simple Policy Based Access Control (PBAC) model originated from the ITU-T Access Control Framework (X.812-ACF) defined in the standard T-REC-X.812-199511 [4] and actually compliant with the Generic AAA AuthZ Framework [1, 2]. The suggested PBAC model consists of two major modules a Policy Enforcement Point (PEP), typically located at the Network Node/Element (such as an OXC), and a Policy Decision Point (PDP) that may be placed together with the PEP or separately depending on the complexity and frequency of events that require policy decisions. A PDP may use different local and external policy repositories often referred as Policy Authority Point (PAP). The UNI specification also suggests using the COPS protocol as policy decision enforcement mechanism if an external PDP is used.

The UNI specification describes few example policies applicable to optical network connection provisioning: Time-of-day based provisioning, identity and credit verification for the connection requestor, and usage based accounting. Although these example can be considered as typical and basic, their implementation requires more complex functionality and infrastructure than just simple PEP-PDP interaction. This is especially related to providing support to access control session/state management [9], the COPS policy enforcement mechanism [13, 14], and usage control [15].

Recent developments in the GAAA-AuthZ framework for Complex Resource Provisioning (CRP) [7] and Token Based Networking (TBN) [16] provide a number of solutions and mechanisms to resolve the problems mentioned above such as:

- AuthZ session management to support complex AuthZ decision and multiple resources access, including multiple resources belonging to different administrative and security domains.
- AuthZ tickets with extended functionality to support AuthZ session management, delegation and obligated policy decisions.
- Authorisation and reservation tokens as policy enforcement mechanisms that can be used in the G2MPLS Control plane and in-band.
- Policy Obligations Handling model to support usable/accountable resource access/usage and additionally global and local user account mapping widely used in Grid based applications and supercomputing.

When considered for supporting Grid resources access over G.OUNI, an AuthZ service may be required to support also AuthZ session termination as a part of the resource release process/sequence.

Although listed above functionalities can be implemented under extended PEP or PDP functionality, such an approach would significantly limit AuthZ service flexibility and potentially affect interoperability of different implementations as the discussed functionalities require agreement on a number of protocols, messaging formats and attributes semantics.

The solution proposed in the GAAA-AuthZ framework is based on using such structural components and solutions as the Token Validation Service (TVS), Obligation Handling Reference Model (OHRM), and XACML policy profile for OLPP. Detailed descriptions of these functionalities and their structural components are provided in Chapter 4.

3.3 GEANT Authorisation and Authentication Infrastructure (eduGAIN)

The Phosphorus AAA/AuthZ infrastructure for multidomain Optical Network Resource Provisioning will need to interoperate with other network domains to ensure cross-domain network resource provisioning. One important component of such cross-domain interoperability is user and resource attribute sharing that in general may include the following aspects: Using common attributes semantics or providing attributes mapping, using common PKI hierarchy or bridging trust relations between interacting domains, using a common message format or providing a request/response gateway, a cross-domain state or session management whose function can also be attributed to the cross-domain gateway.

Most of the Phosphorus testbeds and partners are using the GEANT Education and Research Network infrastructure that provides connectivity for many Research and Education organisations in Europe and also offers connectivity to USA and worldwide. This is essential for the Phosphorus AAA/AuthZ infrastructure to ensure compatibility with the GEANT Authentication and Authorisation infrastructure (AAI) being developed as an eduGAIN (GEANT Authorisation INfrastructure for the research and education community) framework [17, 18]. This section will provide short information about the eduGAIN architecture, components and integration/interoperation issues.

The eduGAIN architecture ensures safe and trustworthy communication between the resource owner (remote domain) and the users home institution (home domain) assumingly belonging to different federations. This is achieved by using a common Metadata Service (MDS), a common PKI hierarchy and a set of naming conventions for all architecture components. The basic idea behind the proposed eduGAIN architecture is to provide interoperability between the existing AAI of the participating NRENs, in particular by interfacing with the following AAI frameworks: Shibboleth (Internet2), PAPI (RedIRIS), Liberty Alliance/FEIDE (UNINETT) and A-Select (SURFnet).

The proposed eduGAIN architecture is based on a set of components which use the trust links established among them by the eduGAIN federation schema (see Fig. 3.4):

The Common eduGAIN services provide common metadata services for the eduGAIN infrastructure and can be requested by Bridge Elements (BE) using defined in the eduGAIN protocols. The current version of the architecture implements only one basic service: The Metadata Service that provides metadata about eduGAIN interfaces, mainly used to locate the appropriate identity repository in the home domain.

The eduGAIN Federation Peering Points (FPP), available in both domains, publish metadata about a federation through the Metadata Service (MDS). Each federation connected to eduGAIN should maintain an own FPP that collects information about the state and changes in all BEs within the federation. This allows each federation to announce its policies via the MDS and keep other participants informed of potential changes.

The Bridging Elements (BE) establish appropriate trust links among AAI components and user applications, adapt syntax, semantics and procedures between eduGAIN and other established infrastructures and individual sites. BEs operate as trust entities/anchors when crossing federation or site boarders. They have pre-established trust relations with the eduGAIN PKI hierarchy and need to be incorporated into the local trust domain. A BE may act as a single Local Federation Adaptor (LFA) that bridges trust relations and other interactions for the whole local federation, or can act as a Local Adaptor (LA) for individual sites or systems.

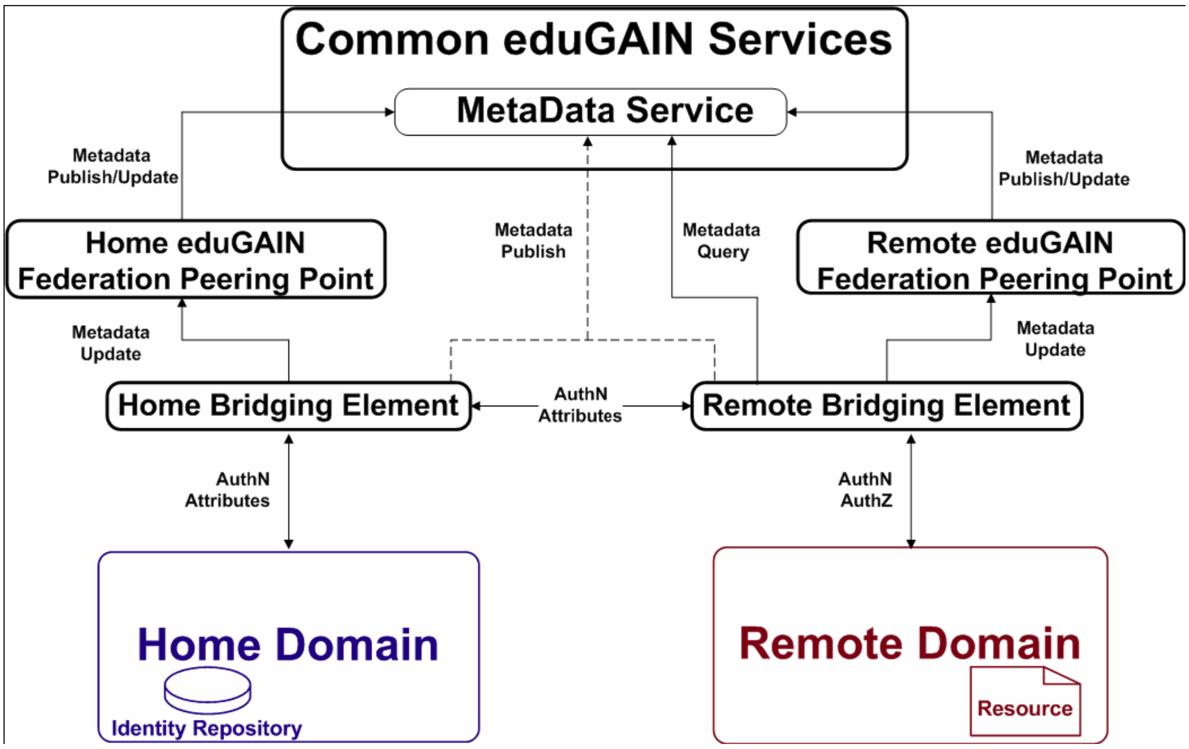

Figure 3.4. Basic components of eduGAIN and their relationship [17]

The eduGAIN protocol specifies the following interactions between the user and the resource:

- Authentication of the user (in the Home Domain), initiated by the Service Provider (in the Remote Domain).
- Attribute request, initiated by the Service Provider to obtain information about the user.
- Authorization for the service, using information about the user provided in the request and/or additionally obtained by the Remote Domain.
- The Metadata Service is used by Federation Peering Points to locate the appropriate interfaces where the above interactions can be performed and to establish trust between them.

The eduGAIN Metadata service uses own namespace "`urn:geant:edugain`" for URN identifiers:

`urn:geant:edugain:component(:mds | :fpp | :be | sp | :ipd |)` – for components `urn:geant:edugain:component:assertion` – for assertions (SAML based)

`urn:geant:edugain:component:protocol` – for protocol (SAML based)

All eduGAIN protocols are SAML based and at the current stage conforming to the SAML1.1 specification. eduGAIN uses standard X.509 PKI certificates but defines a number of mandatory extensions related to key usage and eduGAIN specific PKI policy.

The Java based eduGAIN API uses OpenSAML and Shibboleth libraries and consists of three Java packages that support the following basic services:

- eduGAINVal, providing the services required to carry out the eduGAIN trust validation procedures, and to prepare the exchanged material according to those procedures.
- eduGAINMeta, providing the services required to publish and retrieve eduGAIN metadata.
- eduGAINBase, providing the implementation of the eduGAIN abstract service definition plus specific interfaces for each of the defined profiles.

The following steps should be performed to connect to eduGAIN (useful to know for possible GAAA-AuthZ integration into eduGAIN) [18]:

- Establish an own identity federation. BEs produced along the development of eduGAIN are available for A-Select, PAPI, Shibboleth and the Sun Federation Suite.
- Obtain a federation identifier from the eduGAIN Naming Registry. All component identifiers of the elements connecting the new federation to eduGAIN must use this assigned namespace prefix.
- Integrate the new federation in the eduGAIN trust fabric. This can be done either by accrediting the federation CA to be signed by the eduGAIN CA, or by applying for certificates through eduGAIN CA.
- Design the structure of the eduGAIN connection. It is recommended to start with a highly centralised schema, with a single FPP and a single BE acting as Local Federation Adaptor as the unique points of contact between the new federation and eduGAIN.
- Validate the eduGAIN connection by means of the eduGAIN Validation Facility.
- Start collaboration with the rest of eduGAIN participants in maintaining and enhancing the infrastructure.

3.4 General AAA/AuthZ Requirements for multidomain on-demand network resource provisioning

This paragraph will summarise the various security requirements, which follow from the above described target use cases and are needed to allow inter-domain OLPP. The use of terms MUST, SHOULD and MAY are in accordance with RFC2119.

R1.Authenticatio n	Authentication (AuthN) is the first stage in access control. It is performed to establish a trusted electronic identity of the requesting user. The user MUST present credentials, which has been issued by a person or an organisation which MUST be trusted to check a persons identity according to pre-established procedures (e.g. check identity based on a government issued photo ID and/or credentials from other recognised and trusted registries).
R1.1	An AuthN SHOULD yield a result in the form of: 1) An explicitly provided AuthN ticket or token. 2) An implicit granted access to the protected resource or system. In the latter case the AuthN is confirmed by the start of a session under a users personal- or group ID, that can be supported by system variables.
R1.2	In a multi-domain scenario, only the User Home Organisation SHOULD provide the authentication service. In such case, additional security services MUST provide inter-domain user identity, credentials and attributes translation.
R1.3	In a multi-domain scenario, the (initial) user authentication in a User Home Organisation (UHO) SHOULD be allowed to use a user-centric Trust Anchor (TA), with the user as a root of trust for all following identity translation and attribute management operations. This SHOULD therefore be considered as the most sensitive procedure/operation. However, IdM/IDP or Authorisation (access control) services MAY also verify and request confirmation of the initial user AuthN at the request evaluation or service access stages.
R1.4	In case of using a more extended functionality with Identity management, AuthN SHOULD be allowed to be a basis for issuing proxy and delegated user identity credentials and/or user attributes
R2. Identity Management	Identity management MAY be used as an additional step in Access Control after Authentication and before Authorisation to provide: 1) Single Sign-On (SSO) service in environment with multiple identities (of the same user/requestor) and also multiple domains. 2) flexible user attributes management bound to his/her identity. 3) manage and provide context to user federations and associations, 4) enable user identity delegation both in single domain and multiple domains. Note. A Virtual Organisation (VO) management system MAY be considered as a part of the general Identity Management.
R2.1	Within multi-domain scenario's, each domain MAY contain an Identity Management service (IdM) as to provide: 1) inter-domain or inter-organisational identity translation. 2) independent management of domain’s users and resources membership,

	i.e. associations and federations, 3) possibility to support creation of dynamic security associations (like VO), 4) (user-centric) inter-domain trust management. Note. This functionality can be abstracted to the Security Token Service (STS) as a generic service.
R2.2	An IdM service SHOULD be allowed to issue user credentials (that can be both a user Id and attributes) based on user AuthN or other form of identity credentials. The IdM SHOULD rely on existing trust relationship with AuthN service or other IdM services. There MAY be different models for trust management when issuing identity credentials. 1) The IdM service in a UHO domain MAY rely on existing trust relations between AuhtN services and IdM, e.g. having the same root CA. 2) An IdM service in a remote domain MAY use a direct or indirect trust relationship between UHO AuthN or IdM. Special (business/provisioning) agreements between interacting domains SHOULD define the acceptance policies for remote AuthN or Id credentials. In particular, the acceptable strength of AuthN, or the acceptable chain of trust/credentials, and the Identity delegation conditions (e.g., limited delegation, or full impersonation). 3) Federations or associations in which a user has a proven membership, that are supported by a special membership services such as the VO Membership Service (VOMS), MAY be used for inter-domain attribute- and trust management.
R3. Authorisation	The Authorisation function protects a resource by defining and enforcing access control policies. Authorisation is based on the authenticated identity of an authenticated user or requestor. The identity is represented explicitly in a form of AuthN or Id credentials, that are issued by a trusted AuthN or IdM service. An authorisation service evaluates a request for a resource containing user or requestor credentials according to the resource domain's AuthZ policy, which defines access control rules based on user attributes (group membership, roles or other capabilities).
R3.1	User attributes MAY include user membership attributes from an association or federation that governs network usage, security or imposes resource consumption constraints. During the policy evaluation, an AuthZ service MAY therefore request additional information such as: 1) more user specific credentials, 2) budget related or accounting type of information 3) confirmation information from a users security services provider, authorities, resource managers etc.
R3.2	An AuthZ service MAY include one or more of the following functional modules: 1) A PEP – Policy Enforcement Point

	2) A PDP – Policy Decision Point 3) A PAP – Policy Authority Point
R3.3	When operating in an inter-domain, multi-domain provisioning scenario, an AuthZ service MAY request evaluation of some part of a request by a different AuthZ service, possibly located in another domain. However, in order to protect the integrity of an AuthZ decision, the final composition of the decision MUST be performed by the PDP that received the original request.
R 3.4	Based on a successful authorisation, the AuthZ service MAY issue an AuthZ ticket that MAY be used in subsequent AuthZ requests or MAY be used by the ICC as a base for issuing a reservation ticket. It is essential that,, when presenting AuthZ tickets (or tokens), the ticket or tokens authenticity and integrity within subsequent requests MUST be evaluated by a resource's PEP. For this, the PEP MUST have a secure trust relationship with the PDP in order to exchange the corresponding key material.
R 3.5	An AuthZ service MAY operate in a pull or push mode, or support more complex combinations. Note. It is perceived that the push model will require using AuthZ sessions credentials if the form AuthZ tickets obtained in advance from the resource's AuthZ service or other trusted AuthZ service, e.g. belonging to a VO or other user and resource federation.
R 3.6	An AuthZ service MAY issue provisional authorizations during the reservation stage. Authorizations MAY be altered or made more specific during the access/consumption stage. This requirement MAY also imply evaluation of different criteria and applied policies during the reservation and access stage. E.g. a reservation request may specify only basic requirements towards the resource. Only during the resource allocation phase, a user/application will expect confirmation from the particular resource, which MAY also imply that a different set of user attributes are required to be offered.
R 3.7	Policy and consequently PDP decision may apply special requirements or require special actions that must be taken at the time of the reserved resource access. This can be achieved with so-called policy obligations, originally proposed as a part of XACML policy definition.
R 3.8	Mutual AuthZ MAY be required, E.g. the receiver first asks the sender to receive certain information. Subsequently, when ready, the sender explicitly asks permission from the receiver to send. Applications within the medical- or banking area, are likely to pose such requirements.
R 3.9	AuthZ service SHOULD support AuthZ session management functionality as a part of the complex resource AuthZ and for general performance optimisation.
R 3.10	AuthZ service that supports AuthZ session management may be required to support also AuthZ session termination.
R4. Attribute management	User (and resource) attributes MAY be managed separately by Attribute Authorities (AA) but still in conjunction with the user or resource identities.

	Attribute management MAY be delegated to an association or federation membership service, such as a VOMS [19] in Grid applications or InCommon Federation in Internet2 Shibboleth infrastructure [20, 21].
R 4.1	One of the specific AA infrastructure (AAI) functions is the management of attribute namespaces that are shared between interacting members or domains, or can be mapped/translated by IdM services. For this purpose, the AAI SHOULD provide potentially mapped attributes/namespaces that are directly understood by IdM services or can be mapped (based on known/pre-established relations).
R 4.2	The validity and trustworthiness of attributes will have effect on an AuthZ decision's trustworthiness and MUST therefore be considered in the overall trust-relationship analysis.
R 4.3	Different strength of the user ID and attribute confirmation MAY be required during two-stage reservation and access sequence.
R5. Trust management	All security related operations and resource allocation operations MUST be based on established and traceable trust relations based on mechanisms such as shared secrets, PKI, SPKI, IKE, XKMS, SSL, DAA (as defined in the Trusted Computing Platform architecture [22]), etc.
R5.1	Trust relations, being instant for any particular service invocation, can be invoked dynamically, however SHOULD rely on more static pre-established relations that can be used for initial trust introduction. For example, use published service public key to initiate session to exchange more secure credentials, etc.
R 5.2	A VO MAY be used for creating dynamic security associations of users and resources and will support inter-domain/inter-organisational trust management by providing trust anchor for inter-domain credential management.
R 5.3	DNSSEC MAY contain a VO's or Federation's public key bound to the domain name and MAY be used for user/originator attributes verification and/or initial trust introduction.
R 5.4	All security valid decisions, e.g. delegation, AuthZ or reservation, and credentials MUST have an unbroken and auditable chain of trust.
R6. Federation management	Inter-domain/multi-domain scenarios require some form of federation to be established for user identity-, attribute- and trust management.
R 6.1	Federations that MAY be used for OLPP are inter-university federations like Internet2 InCommon, or VO's maintained by major Grid consortia such as EGEE, OSG etc. In the particular case of inter-domain trust management, such federations SHOULD be useable for attribute management and/or trust management.

R 6.2	Federations, such as a Grid VO, SHOULD be allowed to provide a communication context for services and applications interacting through (enterprise) firewalls.
R7. AuthN/AuthZ service API	AuthN/AuthZ services API (GAAAPI) is required to provide easy integration of AAA/AuthZ services into NRPS/CRP applications. GAAAPI flexibly and dynamically request AuthN, AuthZ and Attribute services from network services and applications.
R 7.1	GAAAPI SHOULD define protocols, request- and response message formats, basic commands and extensibility procedure, basic configuration profiles, namespace resolution/management and enumerated attribute values assignment.
R 7.2	GAAAPI SHOULD allow easy integration with existing Grid and Web Services middleware.
	GAAAPI SHOULD allows easy and flexible configuration of trust relations between interacting AAA/AuthZ and NRPS components in a multidomain infrastructure using both hierarchical PKI and mutual PKI based trust relations

Note. Requirements to the Token Validation Service (TVS) are discussed in section 4.3. Although current TVS implementation is provided as a part of the general GAAAPI package, TVS can be also used separately with different AAA/AuthZ service implementations.

4 AAA/AuthZ architecture for Optical Network Resource Provisioning

This chapter describes a GAAA/AuthZ architecture for Optical Network Resource Provisioning as a particular use case of the general Complex Resource Provisioning (CRP). Further it provides a general abstraction of the CRP operational models and describes in detail the major components such as AuthZ ticket and token formats, XACML policy profile for ONRP, policy obligations and the Obligations Handling Reference Model. Also the Token Validation Service (TVS) model and operation is explained in detail as it is considered as an important component of the WP4 AAA/AuthZ architecture to provide flexibility in policy enforcement for the reserved network resources at the network access stage.

4.1 CRP operational models and AAA Authorisation service architecture

Two major use cases for the general CRP [6, 7] are network on-demand provisioning systems that are using ONRP and Virtualised Collaborative Environments (VCE). Although different in current implementations, they can be abstracted to the same CRP operational model when considering their implementation with the SOA based Grid or Web Services. This abstraction is considered as an important aspect to provide a common basis to integrate and provide a common access control infrastructure for dedicated optical networks and Grid resources accessed and brokered over the network.

The typical on-demand resource provisioning includes two major stages: resource reservation and the reserved resource access/consumption. In its own turn, the reservation and allocation stage includes four basic steps: (1) resource lookup, (2) complex resource composition (including alternatives), (3) reservation of individual resources and their association with the reservation ticket/ID, and (4) finally delivery or deployment/allocation. The reservation stage may require the execution of complex procedures that may also request individual resources authorisation. This process can be controlled by the AAA driving policy that should support the whole provisioning workflow and related AuthZ policy, or driven by a Meta Scheduling system.

In the discussed CRP model, domains are defined (as associations of entities) by a common policy under single administration, common namespaces and semantics, shared trust, etc. In this case, the domain related security context may include: namespace aware names and ID's, policy references/ID's, trust anchors (TA), authority references, and also dynamic/session related context [9]. For the generality, domains can be hierarchical, flat or organized in the mesh, but all these cases require the same basic functionality for the access control infrastructure to manage domain and session related security context.

CRP for the hierarchical and distributed resource management model requires the following functionality from the GAAA-AuthZ infrastructure:

- multiple policies processing and combination.
- attributes/rules mapping/converting based on inter domain trust management infrastructure.
- hierarchical roles/permissions management, including administrative policies and delegation.
- policy support for different logical organisation of resources, including possible constraints on resource combination and interoperation.

Figure 4.1 illustrates major interacting components in the multi-domain CRP using ONRP as a major use case:

- A User/Requestor.
- A Target end service or application.
- Multiple Network elements (NE) (related to the Network plane).
- Network Resource Provisioning Systems (NRPS) (typically related to the Control plane).
- AAA service controlling access to the domain- related resources that can also operate own communication infrastructure, including components PEP, PDP, PAP.
- Token Validation Services (TVS) that allow efficient authorisation decision enforcement when accessing reserved resources.

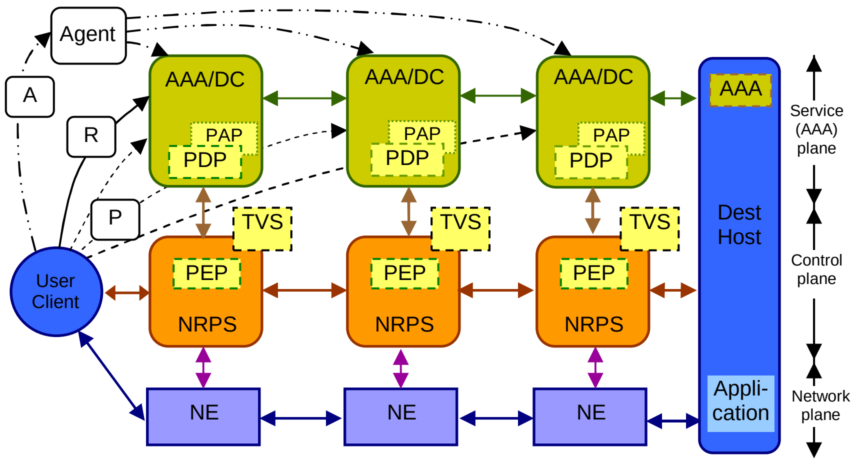

Figure 4.1. Components involved in complex resource provisioning and basic sequences (agent, relay, and polling)

The above described CRP model can be generalized for another typical CRP use case of the Virtual Laboratory (VL) workspace provisioning if we consider virtual Workspace elements (WSE) in the hierarchical VL organisation as separate resource domains that can be logically organised into different structures and described with the same attribute types as traditional network domains.

Figure 4.1 also illustrates different provisioning models or sequences that can be executed when composing a complex resource:

Polling sequence when the user client polls all resource or network domains, builds the path and makes the reservation.

Relay or hop-by-hop reservation sequence (also referred as chaining sequence) when the user contacts only the local network domain/provider providing destination address, and each consecutive domain provides a path to the next domain.

Agent sequence when the user delegates network provisioning negotiation to the agent that will take care of all necessary negotiations to provide required network path to the user. A benefit of outsourcing resource provisioning is that the agents can maintain their own reservation and trust infrastructure.

Access to the resource or service is controlled by the NRPS and protected by the AAA service that enforces resource access control policy by placing a Policy Enforcement Point (PEP) gateway at the NRPS (cp chapter 3.1). Depending on the basic GAAA-AuthZ sequence (push, pull or agent) [2, 3], the requestor can send a resource access request to the resource or service (which in our case are represented by NRPS) or an AuthZ decision request to the designated AAA server which in this case will act as a Policy Decision Point (PDP). The PDP identifies the applicable policy or policy set and retrieves them from the Policy Authority Point (PAP), collects the required context information and evaluates the request against the policy.

The user can provide as much (or as little) information about the subject/requestor, the resource or the action as it decides necessary according to the implemented authorisation model and resource access control policies. The Policy Decision Point (PDP), which is the part of the AAA AuthZ service, evaluates the request and makes the decision whether to grant access or not. Based on a positive AuthZ decision (in one domain) the AuthZ ticket (AuthzTicket) can be generated by the PDP or PEP and communicated to the next domain where it may be processed as a security context or policy evaluation environment.

It is essential in the Grid/Web services based service oriented environment that AuthZ decision must rely on both Authentication (AuthN) of the user and/or the request message and Authorisation (AuthZ). AuthN credentials are presented as a security context in the AuthZ decision making.

In order to get access to the reserved resources the requestor needs to present the reservation credentials that can be in a form of an AuthZ ticket or a token (AuthzTicket or AuthzToken) which will be evaluated by the PEP to grant access to the reserved network elements or the resource. In more complex provisioning scenarios the token or credential validation function may be outsourced to the TVS service that can additionally support a interdomain trust management infrastructure for off-band token and token key distribution between the PEP-NRPS and AAA services. Token and token key generation and validation model can use either shared secret or PKI based trust model. The TVS as a special GAAA-AuthZ component to support token-based enforcement mechanism in the Token Based Networking (TBN) is described below. TVS can be implemented as a proprietary AAA-NRPS solution or it can use one of the proposed standard models of the Credential Validation Services (CVS) [23] or WS-Trust Secure Token Service (STS) [24].

Using AuthZ tickets during the reservation stage for communicating the interdomain AuthZ context is essential to ensure effective decision making. At the service access/consumption stage the reserved resource may be simply identified by the reservation ID created as a result of the successful reservation process. To avoid significant policy enforcement overhead when handing service reservation context, the ticket can be cached by an NRPS or a TVS in each domain and referred to with the AuthzToken that can be much smaller and even communicated in-band. At the resource PEP it can be compared with the cached AuthzTicket, AuthZ session context or reservation context and will allow local PEP/resource access control decisions. Such an access control enforcement model is being implemented in the Token Based Network (TBN) described in section 6.

Figures 4.2 and 4.3 below illustrate how the general CRP operation models can be implemented for real multidomain optical network provisioning use cases. They depict two approaches: (1) service provider approach that uses either chained signalling (ingress domain will act as agent to next domain) or tree signalling what corresponds to the generic relay and polling sequences as discussed above; and (2) Grid approach where all network domains are represented as Grid resources and interdomain signalling is provided by the Grid-enabled Network Service Plane service. This allows using standard Grid scheduling system, such as Viola Meta Scheduler that uses Web Services Agreement (WSAG) based negotiation model and framework [25, 26], to schedule/reserve both computing resource and the whole end-to-end network path/connectivity. Both approaches can use token-based policy enforcement mechanism, which operates at the network control plane level, to secure user access to the reserved network resource. In the chain model, the token key generation can be done within each domain to indicate commits of the service. In the tree model, the token key is generated in the last domain, however this model can use different options as applied to token generation and handling.

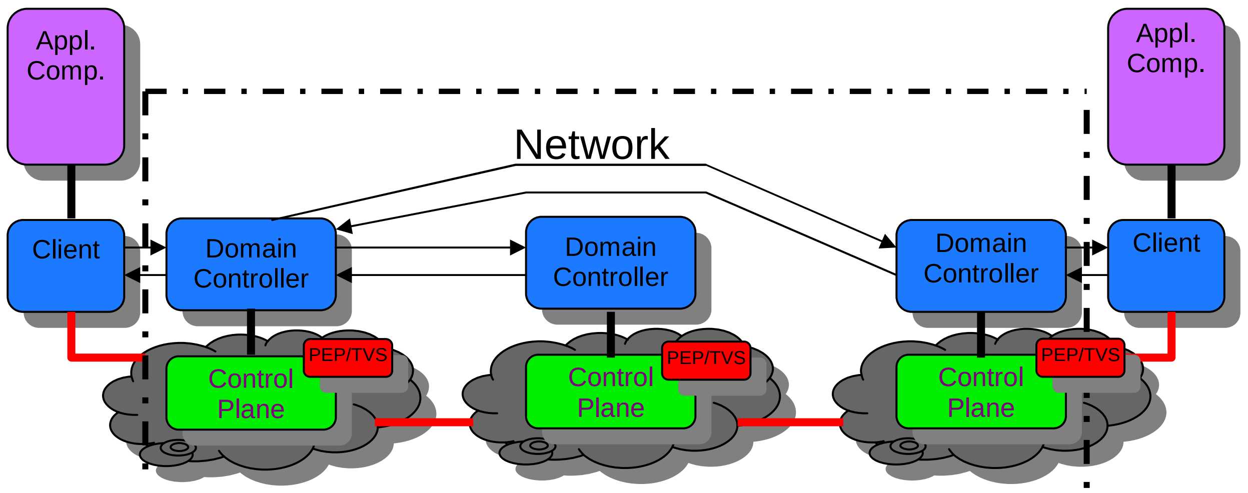

Figure 4.2. Provider approach in ONRP.

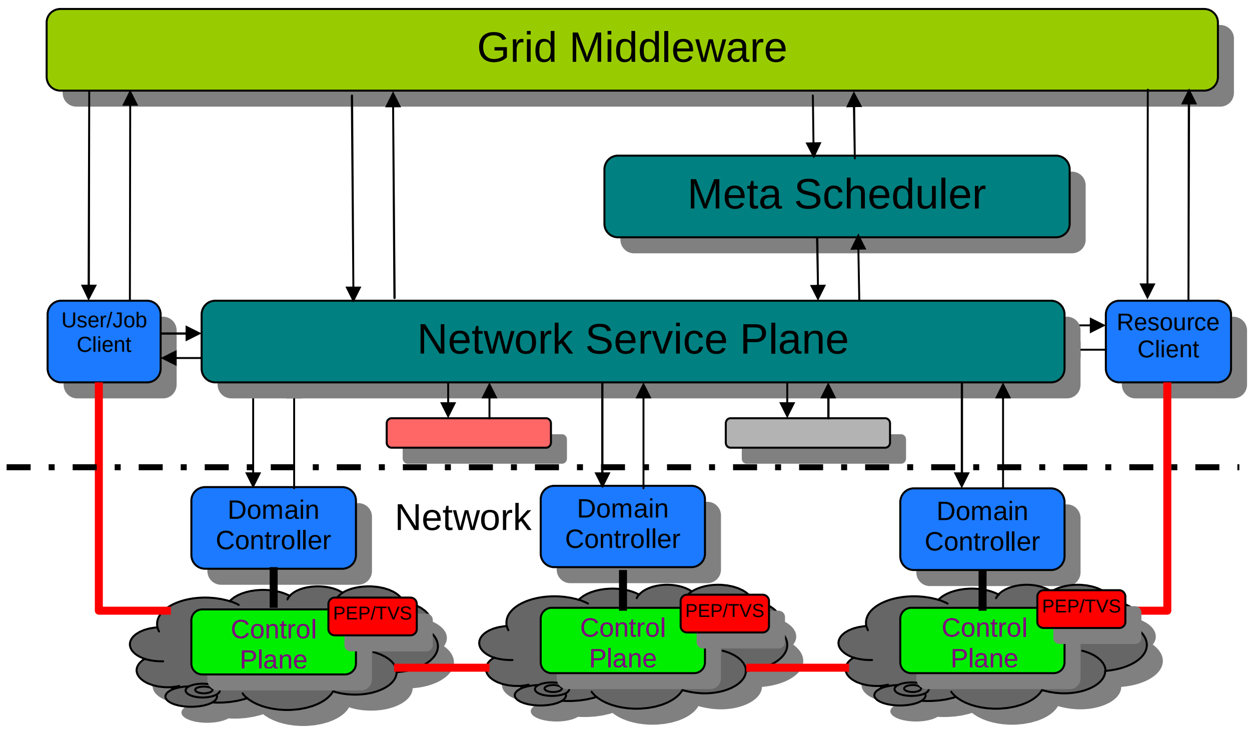

Figure 4.3. Grid approach in ONRP.

4.2 GAAA-AuthZ access control mechanisms and components

4.2.1 GAAA-AuthZ access control mechanisms and components overview

The proposed GAAA-AuthZ access control mechanisms and components extend the generic model described in GAAA-AuthZ with the specific functionality for on-demand ONRP, in particular:

- AuthZ session management to support complex AuthZ decision and multiple resources access, including multiple resources belonging to different administrative and security domains.
- AuthZ tickets with extended functionality to support AuthZ session management, delegation and obligated policy decisions.
- Authorisation and reservation tokens as policy enforcement mechanisms that can be used in the G2MPLS Control plane and in-band.
- Policy Obligations Handling model to support usable/accountable resource access/usage and additionally global and local user account mapping widely used in Grid based applications and supercomputing.

When considered for supporting Grid resources access over G2MPLS G.OUNI (see section 3.2 and [12, 13]), the AuthZ service may also require to support AuthZ session termination as a part of the resource release process/sequence. However at the moment this issue is not considered.

Although the above listed functionalities can be implemented under extended PEP or PDP functionality, such an approach would significantly limit AuthZ service flexibility and potentially affect interoperability of different implementations as the discussed functionalities require an agreement on a number of protocol issues, messaging formats and attribute semantics.

The solutions proposed in the GAAA-AuthZ framework are based on using such structural components and solutions as a Token Validation Service (TVS), the Obligation Handling Reference Model (OHRM), and the XACML policy profile for OLPP, all discussed in this section.

Figure 4.4 illustrates the major GAAA-AuthZ modules and how they interact when evaluating a service request.

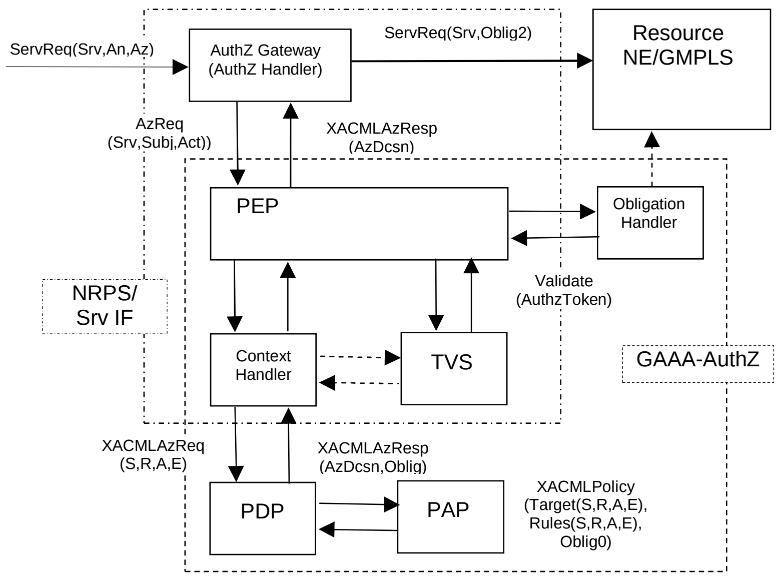

Figure 4.4. GAAA-AuthZ components providing service request evaluation

The authorisation service is called from the service/application interface via the AuthZ gateway (that can be just an interceptor process called from the service or application) that intercepts a service request ServiceRequset (ServiceId, AuthN, AuthZ) that contains a service name (and variables if necessary) and AuthN/AuthZ attributes. The AuthZ Gateway extracts necessary information and sends an AuthZ request AuthzRequest (ServiceId, Subject, Action), that contains a service name ServiceId, the requestor's identification and credentials, and the requested Action(s), to the PEP. The major PEP's task is to convert AuthZ request's semantics into the PDP request which semantics is actually defined by the used policy. When using an XACML policy and correspondingly an XACML PDP, the PEP will send an XACML AuthZ request to the PDP in the format (subject, resource, attributes, (environment)). If in general case the XACML policy contains obligations, they are returned in the XACMLAzResponse (AuthzDecision, Obligations). The PEP calls the Obligation Handler to process obligations which are defined as actions to be taken on the policy decision or in conjunctions with the service access (like account mapping, quota enforcing, logging, or accounting).

If the service request contains an AuthZ token that may reference a local or global reservation ID, or just identifies an AuthZ session in which context the request is sent, the token validation is performed by the Token Validation Service (TVS). The TVS is typically called from the PEP and returns a confirmation if the token is valid. Separating TVSs as a separate function or service allow creating flexible token and/or ticket policy enforcement infrastructures for on-demand network resource provisioning.

4.2.2 AuthZ Ticket and Token formats for extended AuthZ Session Management

4.2.2.1 *AuthZ Ticket use in GAAA-AuthZ*

The authorisation ticket (AuthzTicket) is a part of the GAAA-AuthZ framework functionality and allows the transfer of a full AuthZ decision and policy enforcement context between a requestor and an AuthZ service or between different AuthZ/security domains. More general information about using AuthZ tickets in complex AAA operational model is provided in chapter 2.

As discussed above, there are two types of sessions in the proposed CRP model that require a security context management: reservation and provisioning session, and the reserved resource access session. Although the provisioning session may require wider security context support, both of them are based on the (positive) AuthZ decision, may have a similar AuthZ context and will require a similar functionality when considering distributed multi-domain scenarios. In this case an AuthZ ticket should provide all necessary context information and will serve as a session or access credentials.

To reduce possible high communication and processing overhead because of a potentially large size of AuthZ ticket, an AuthZ token can be used. In this case the AuthZ token should unambiguously reference the original AuthZ ticket or instant AuthZ session context that must be securely stored at the resource or access point. At the time of the authorised or reserved resource access, the original AuthZ ticket or AuthZ session context object will be retrieved and used for the request evaluation. When used together, AuthzTicket and AuthzToken share the SessionId attribute which can be either global or local reservation/session ID and are cryptographically connected, e.g. the token value is a hash value of the ticket content. An AuthzTicket must be digitally signed to keep its integrity.

In a particular use case of the Token Based Networking, the AuthzTicket is used for programming a TVS and provides both a reservation ID/reference and detailed information for configuring a TBS (token based ForCES switch).

4.2.2.2 ***AuthzTicket data model and schema***

An AuthzTicket has a complex but flexible format. The current AuthzTicket format and its implementation in the GAAAPI support extended functionality for distributed multidomain hierarchical resources access control and user roles/permissions management, in particular, administrative policy management (as defined in XACML 3.0 Administrative policy profile [27]), capabilities delegation and conditional AuthZ decision assertion (to support XACML policy obligations). It is one of the general design suggestions that an AuthzTicket should be easy to map to the SAML AuthzDecision Assertion [28] or to XACMLAuthzDecision Assertion defined by the SAML profile of XACML [29, 30].

Diagrams 4.5 and 4.6 below illustrate the top and main AuthzTicket elements and the resource element structure for the ForCES token-based switch (TBS). The AuthZ ticket core schema and TBS profile schema are provided as part of the GAAAPI package and are available from the Phosphorus WP4 Wikipage. Note, the resource element is defined as an extendable in the AuthzTicket schema what allows to incorporate with different types of target resources (to which AuthZ decision is applied).

The AuthzTicket contains the following major groups of elements that allow for extended AuthZ session security context management:

- The Root element attributes `TicketID`, `SessionID`, and `Issuer` that allows for the ticket unique identification and defines its binding to the session and domains related processes/authorities.
- The `Decisions/Decision` element that holds the PDP AuthZ decision bound to the requested resource or service expressed as the `ResourceID` attribute.
- The `Resources` extendable element that may hold proprietary description of the reserved resource.
- The `Actions/Action` complex element contains actions which are permitted for the subject or its delegates.
- The `Subject` complex element contains all information related to the authenticated subject who obtained permission to do the actions, including sub-elements: `Role` (holding subject's capabilities), `SubjectConfirmationData` (typically holding AuthN context), and extendable sub-element `SubjectContext` that may provide additional security or session related information, e.g. subject's VO, project, or federation.
- The `Delegation` element allows delegating the capabilities defined by the AuthzTicket to another subject(s) or community. The attributes define restriction on type and depth of delegation
- The `Conditions` element specifies the validity constrains for the ticket, including validity time and the AuthZ session identification and additionally context. The extensible `ConditionAuthzSession` element provides rich possibilities for AuthZ context expression.
- The `Obligations/Obligation` element can hold obligations that a PEP/resource should perform in conjunction with the current PDP decision.

The semantics of AuthzTicket elements are defined in such a way that allows easy mapping to related elements in SAML and XACML. The first three elements the `Decision`, the `Actions/Action`, and the `Subject` have a direct mapping to the related SAML elements. Other AuthzTicket elements the `Delegation`, the `Conditions`, and the `Obligations/Obligation` element, which is originated from XACML, can be implemented as extensible element of the SAML `Condition` element.

Note. In the TBS profile, the `SessionID` attribute can hold a GRI, an additional session or domains related information that can be put into the extensible `ConditionAuthzSession` element or into the proprietary structured `Resource` element.

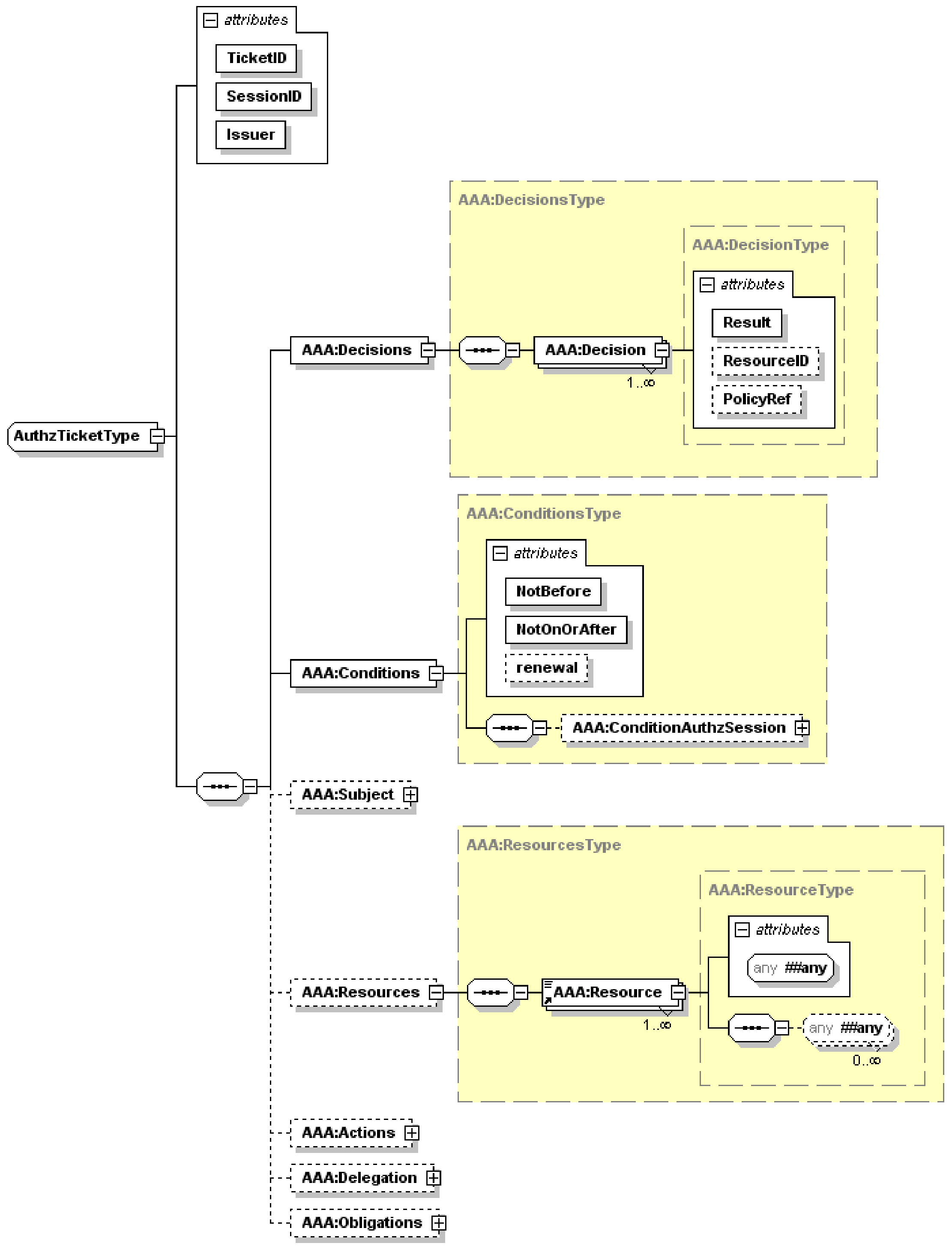

Fig. 4.5 AuthzTicket top and main elements (note, the Resource element is defined as an extendable).

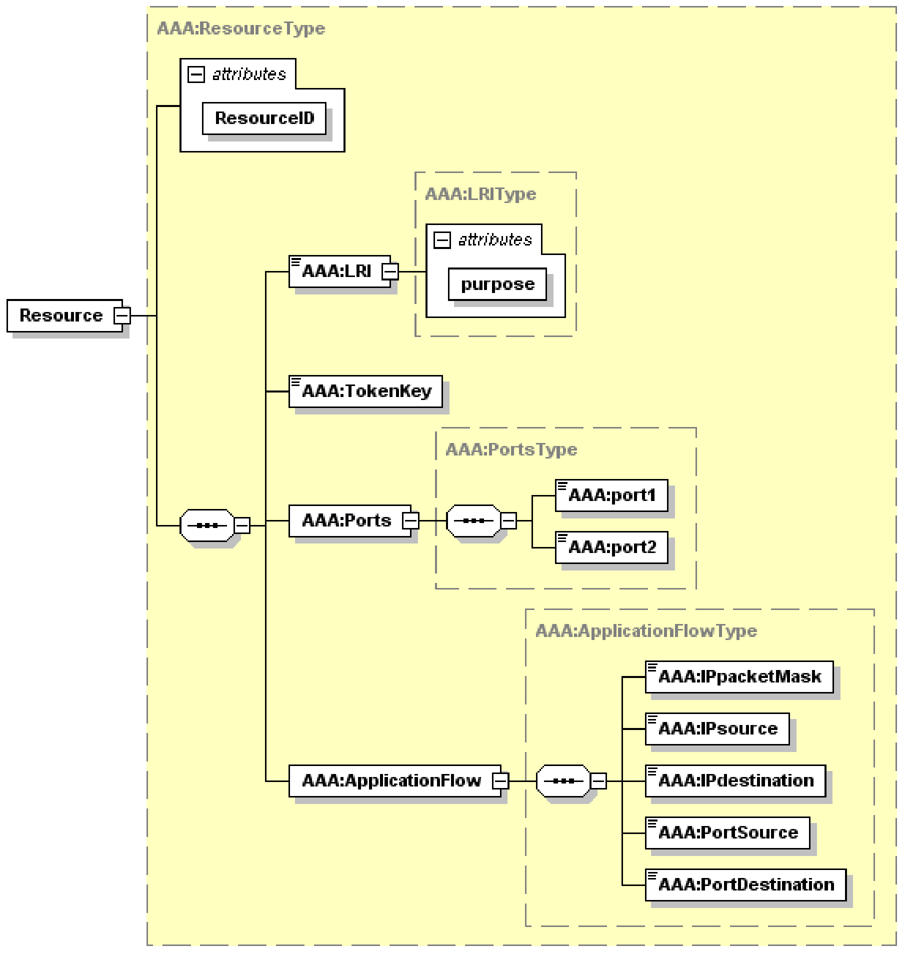

Fig 4.6. The Resource element structure for the ForCES token-based switch (TBS)

The AuthzTicket is digitally signed (as shown in the example) and cached by the resource's AuthZ service. To reduce communication overhead when using an AuthzTicket for consecutive request validation, the associated AuthZ token (AuthzToken) can be generated from the AuthzTicket. The AuthzToken may contain just two elements: `TokenID = TicketID` and `TokenValue = SignatureValue`, needed for the identification of the cached AuthzTicket.

The current AuthzTicket functionality is supported by the GAAAPI package (see chapter 6 for details). Further development will include adding the following additional functionality:

- Elements or attributes that can support mutual AuthZ or session negotiation. This is desirable to have even if the negotiation protocol will have its own messages format, because the user/AuthZ session credentials have to be bound to the requestor/subject credentials and their AuthN context.
- Supporting consumable resource attributes (e.g., usage time, data transferred, number of access), and additionally collecting accounting data.

4.2.3 Tickets and tokens handling in GAAA/AuthZ

Tickets and tokens handling is an important functionality of the GAAA-AuthZ architecture to allow authorisation session security context management in combined agent-pull-push operation models. This functionality is not explicitly defined in the generic X.812 Access Control framework [4] and in the RBAC model [31]. Depending on the implementation this functionality can be attributed to both PEP and PDP. It can not be instantiated to just the request context handling because its operation may result in a definite decision based on local request evaluation against a provided AuthZ ticket without calling a PDP or doing policy evaluation.

In the GAAA-AuthZ, this specific functionality is supported by the Triage module that provides the following functionality:

- Evaluate the request against the provided AuthZ ticket and provide a decision on the requested action or resource.

  Note. In fact, Triage confirms or denies a decision contained in the ticket, although in most cases the ticket will only be issued to positive decision.
- Underlying Triage operations may include: request validation, ticket validation, request classification (to define candidate PDP for processing), etc.

  Note. Such functions in the request (pre-) processing as attributes validation and request typically attributed to the general context handling functionality that may be related to PEP or PDP.

Although the Triage function provides an initial request evaluation, it should be considered as a function called from the PEP (and optionally from PDP in case when a policy suggests evaluating AuthZ context what may be a case for multidomain/complex resource request). This allows splitting the generic PEP function to convert between application specific request semantics and those required by the policy evaluation process.

Under some considerations, the Triage functionality can be attributed to the PDP (or PEP) but as it is discussed above its specific functionality is different from the generic PDP and PEP functionality. Actually, the Triage implementation in the GAAAPI allows calling the Triage function from the PDP or PEP.

Picture 4.7 below illustrates how the Triage interacts with the PEP, the PDP and other generic AAA-AuthZ components.

The following summarises the Triage operation on AuthZ tickets and token handling/evaluation:

- An AuthzTicket is issued by the PDP and MAY be issued by the PEP
- An AuthzTicket MUST be signed to ensure authenticity and integrity
- An AuthzTicket MUST contain all necessary information to make a local PEP-Triage request verification
- When using AuthzTokens, AuthzTickets MUST be cached; a resolution mechanism from token to ticket must be provided

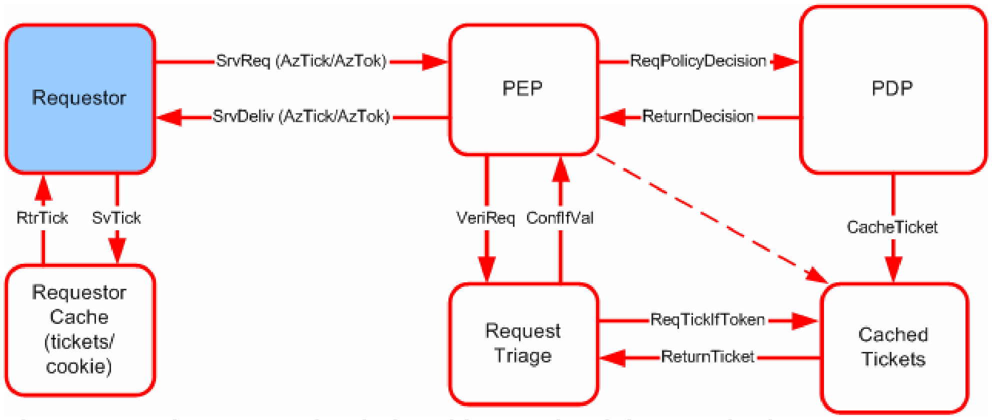

Figure 4.7. Triage operation in handling AuthZ tickets and tokens.

4.2.4 Policy Obligations and Obligations Handling Reference Model (OHRM)

In many applications, policies may specify actions that must be performed either instead of or in addition to the policy decision. In the XACML specification [24], obligations are defined as actions that must be performed in conjunction with policy evaluation on a positive or negative decision. Obligations are included into the policy definition and returned by PDP to PEP which in its turn should take actions as prescribed in the obligation instructions or statements. A more extended obligations definition, taken from the XACML specification, and examples of obligations expression are provided in the Appendix C.

In the context of the GAAA-AuthZ architecture for ONRP, obligations provide an important mechanism for policy decision enforcement in the provisioned network resources, in particular, obligations can be used for mapping global user ID/account to local accounts or groups, assigning quotas or usage limits.

Figure 4.8 below illustrates the Obligations Handling Reference Model (OHRM) for processing obligations in the general case of the Domain-Central AuthZ service (DCAS) that can be part of the Domain Controller (DC). The DCAS means that all domain located resources and services use a central AuthZ service that maintains a common set of policies for this domain. The described processing model is compliant to the model used in XACML (refer to XACML2.0 standard or see Appendix C) but adds Web services and AuthZ callout protocol details and specifically focuses on the obligations handling dataflow.

The obligations handling model allows two types of obligations execution: at the time of receiving obligations from the PDP and at the later time when accessing a resource or performing an authorised action. First type is described below, the second type of handling obligations can be achieved by using AuthZ tickets that hold obligations together with AuthZ decisions.

A number of assumptions are made to reflect possible options in an AuthZ service infrastructure implementation and different types of obligations both stateful and stateless that are concerned with assigning pool accounts, enforcing quotas, controlling usable resource (e.g., number of resource access, purchased video/music listening time, etc.), logging and accounting.

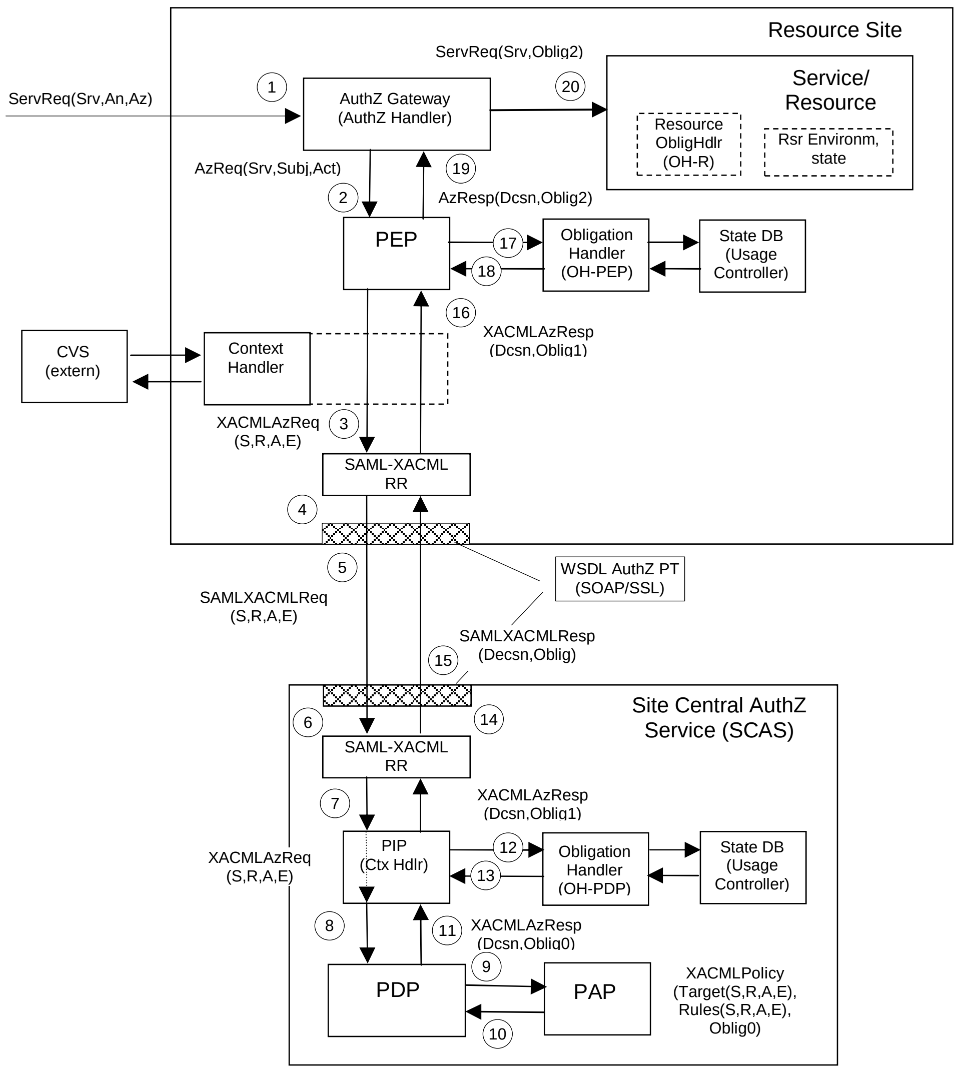
Resource Site
ServReq(Srv,Oblig2)
ServReq(Srv,An,Az)
1
AuthZ Gateway (AuthZ Handler)
20
Service/ Resource
Resource ObligHdlr (OH-R)
Rsr Environm, state
AzReq(Srv,Subj,Act)
19
2
AzResp(Dcsn,Oblig2)
PEP
17
18
Obligation Handler (OH-PEP)
State DB (Usage Controller)
16
XACMLAzResp (Dcsn,Oblig1)
CVS (extern)
Context Handler
XACMLAzReq (S,R,A,E)
3
SAML-XACML RR
4
5
WSDL AuthZ PT (SOAP/SSL)
SAMLXACMLReq (S,R,A,E)
SAMLXACMLResp (Decsn,Oblig)
15
Site Central AuthZ Service (SCAS)
6
14
SAML-XACML RR
7
XACMLAzResp (Dcsn,Oblig1)
PIP (Ctx Hdlr)
12
13
Obligation Handler (OH-PDP)
State DB (Usage Controller)
XACMLAzReq (S,R,A,E)
8
11
XACMLAzResp (Dcsn,Oblig0)
9
PDP
PAP
10
XACMLPolicy (Target(S,R,A,E), Rules(S,R,A,E), Oblig0)

Figure 4.8. Generic authorisation dataflow and obligations handling in distributed AuthZ service.

It is important to notice that obligations are an integral part of the policy and typically included into the policy at the stage of its creation by the policy administrator or resource owner. For the manageability purpose, policy is considered stateless and the statefulness of obligations is achieved by the obligation handlers. The obligations enforcement process can be resulted either in modifying the service request (e.g., map from subject to account name/type) or by changing the resource/system sate or environment.

For the general (stateful) obligations handling process we can distinguish the following stages (note: not all stages are necessary to be implemented in a simple use case but they may exist in different cases):

Obligation0 = tObligation => Obligation1 (“OK?”, (Attributes1 V Environment1))
=> Obligation2 (“OK?”, (Attributes2 V Environment2)) => Obligation3 (Attributes3 V Environment3)

1) Obligation0 – (stateless) obligations are returned by the PDP in a form as they are written in the policy. These obligations can be also considered as a kind of templates or instructions, tObligation. (Important to mention that due to security reason obligations format and semantics should not use executable code or reference to locally executed commands).

2) Obligation1 or Obligation2 – obligations have been handled by the obligation handler at the DCAS/PDP side or at the PEP side, depending on implementation. In this case templates or instructions of the Obligation0 are replaced with the real attributes in Obligation1, e.g. in a form of "name-value" pair. During this stage, the obligation handler can actually enforce obligations or modify obligations and send them further for enforcement by the resource. The result of obligations processing/enforcement, can be returned in a form of modified AuthzResponce (Obligation1) or in a form of global resource environment changes that will be taken into account at the time when the requested service/resource are provided or delivered. In both cases (and specifically in the last case) obligation handler should return notification about fulfilled obligated actions, e.g. in a form of Boolean value "False" or "True", which will be taken into account by PEP or other processing module to finally permit or deny service request by PEP.

3) Obligation3 – this is the final stage when obligations actually take effect, which can be defined as obligations "termination" or "sink". This is done by the resource itself or by services managed/controlled by the resource.

In the proposed model, option with Obligation1 handling stage at the DCAS or PDP side is introduced to illustrate a case when we need to implement a stateful PDP/DCAS what is important for the general CRP and ONRP use cases in particular. However, this should not be considered as XACML specification violation as distinguishing between PEP and PDP functions in the generic obligations handling model is based on what module actually makes policy based request evaluation.

One of the important aspects of the general obligations handling model is not discussed here, namely logical or time wise sequence of enforcing obligations, but this is a topic of recent discussion on the "xacml-dev" and "ogsa-authz" mailing lists.

4.2.5 XACML policy profile for OLPP/CRP

The XACML policy format provides rich functionality for Complex Resource Provisioning in its core specification [30] and special profiles for multiple [31] and hierarchical resources [32], and for RBAC [33]. Hierarchical policy management and dynamic rights delegation, that are considered as important functionality in multidomain ONRP, can be solved with the XACML v3.0 administrative policy profile [27]. XACML allows flexible definition of authorisation rules, conditions and use of different attribute formats.

A XACML policy is defined for the so-called target triad "Subject-Resource-Action" (S-R-A) which can also be completed with the Environment (S-R-A-E) component to add additional context to instant policy evaluation. The XACML policy can also specify actions that must be taken on positive or negative PDP decisions in the form of an optional Obligation element. This functionality is important for the potential integration of the AuthZ system with logging or auditing facilities.

A decision request sent in a request message provides a context for the policy-based decision. The policy applied to a particular decision request may be composed of a number of individual rules or policies. Few policies may be combined to form a single policy that is applicable to the request. XACML specifies a number of policy and rule combination algorithms. The response message may contain multiple result elements, which are related to individual resources.

Any of the S-R-A-E elements allow an extensible “Attribute/AttributeValue” definition to support different attributes semantics and data types. Additionally, XACML allows referencing (internal) context information (from the request message) and external XML document elements by means of XPath functionality.

Two mechanisms can be used to bind the XACML policy to the resource: a Target element can contain any of S-R-A-E attributes and a policy identification attribute IDRef. XACML policy format that provides few mechanisms to add and handle domain or session related context during the policy selection and request evaluation:

- Policy identification that is done based on the Target comprising of the Resource, Action, Subject, and optionally Environment elements.
- Attributes semantics and metadata can be namespace aware and used for attributes resolution during the request processing.
- AuthZ ticket that can be provided as an environment or resource attribute.

The XACML hierarchical resource profile [33] specifies how XACML can provide access control for a resource that is organized as a hierarchy. Examples include file systems, data repositories, XML documents, and also network resources and Grid jobs executing environment. The profile introduces new resource attributes identifiers that may refer to the “resource-ancestor”, “resource-parent”, or “resource-ancestor-or-self”. There may be different matching expressions for the Resource/Attribute/AttributeValue when using XACML hierarchical resource profile what should allow to create a policy for the required resource hierarchy or other logical organisation.

Such specific use case as multidomain ONRP require that the resource reservation policy in each successive domain will rely on the previous domain positive AuthZ decision and it may also additionally require informing the next domain. In a simple case, this can be achieved by placing an AuthZ or reservation ticket from the previous domain in the Environment element. When the sequence is important it can be achieved with the ordered rules and policies combination algorithms defined for the Policy Set or Policy [8].

The XACML RBAC profile [34] provides extended functionality for managing user/subject roles and permissions by defining separate Permission <PolicySet>, Role <PolicySet>, Role Assignment <Policy>, and HasPrivilegeOfRole <Policy>. It also allows using multiple subject elements to add hierarchical group roles related context in handling RBAC requests and sessions, e.g., when some actions require superior subject/role approval to perform a specific action. In such a way, a RBAC profile can significantly simplify rights delegation inside a group of collaborating entities/subjects which normally would require complex credentials management.

The XACMLv3.0 administrative policy profile [27] introduces extensions to the XACML v2.0 to support policy administration and delegation. This is achieved by introducing the PolicyIssuer element that should be supported by a related administrative policy. The dynamic delegation permits some users to create policies of limited duration to delegate certain capabilities to others. Both of these functionalities are relevant to the hierarchical resources and user roles management in CRP and currently being investigated.

The XACMLv3.0 policy profile can indicate if the policy is issued by the trusted PolicyIssuer for the particular domain. In this case the PDP will rely on an already assigned or default PAP and established trust relations, otherwise when other entity is declared as a PolicyIssuer, the PDP should initiate checking administrative policy and delegation chain what is a suggested functionality of the PIP module.

In order to use the XACML format for AuthZ in ONRP, a special XACML-NRP (or more general XACML-CRP) profile for Network Resource Provisioning can be defined addressing the following issues:

- Namespace definition for the network resources, user attributes, and GAAA/AuthZ components
- Attribute semantics and expression format, including support list of enumerated values, if necessary
- Set of basic rules and policy templates (including possible mapping to existing policies in this domain)

A successful XACML-NRP profile introduction will depend on available reference implementation. The goal of the WP4 AAA development is to provide a reference XACML-NRP implementation on the base of the GAAA Toolkit and GAAAPI that will address the following issues:

- namespace support, attribute formats and semantics, including enumerated values, typically provided by profile related attribute handler or helpers
- support of underlying trust infrastructure/fabric.

Some general examples of the XACML policies are provided in the Appendix D.

4.3 Token Validation Service (TVS)

4.3.1 Basic TVS Functionality

One of the WP4 goals is the development of the effective and reliable policy enforcement mechanisms for on-demand network provisioning. At the Networking and dataflow layers this can be achieved by using reliable mechanisms to cryptographically bind data-flows to their origin or target applications and corresponding enforcement mechanisms when dataflow are sent over heterogeneous network infrastructure consisting of dedicated and public optical networks. Important component of such mechanisms is distribution of the key material between domains at the end-stations and gateway routers that allows creating overlay virtual infrastructure of the provisioned network. The data-flow bindings can be represented by a token embedded into the IP packet stream or included inside control packets. One of the WP4 developments is an IETF ForCES based router that acts as a gateway between a general purpose IP network (e.g. campus network) and a dedicated GMPLS network, enforcing and providing tokens at both the IP and GMPLS side. It will allow the integration of token mechanisms inside GMPLS control plane layer using RFC2750 policy data object as a base

The Token Validation Service (TVS) is a component of the GAAA-AuthZ infrastructure supporting token based policy enforcement mechanism during the user access of the reserved service or network. Basic TVS functionality allows checking if a service/resource requesting subject or other entity, that posses/presents current token, has right/permission to access/use a resource based on advance reservation to which this token refers. During its operation TVS checks if a presented token has reference to a previously reserved resource and a request resource/service confirms to a reservation condition. It is intended that extended TVS functionality will also support policy enforcement for the consumable (or usable) resource.

In a simple/basic scenario, TVS operates locally and checks local reservation table directly or indirectly using reservation ID (typically, Global Reservation Id - GRI). It is suggested that in multidomain scenario each domain may maintain Local Reservation ID (LRI) and its mapping to the GRI

In more advanced scenario TVS should allow creation of a TVS infrastructure to support tokens and token related keys distribution to support dynamic resource, users or providers federations.

TVS functionality should support two basic use cases: Token Based Networking (TBN) using in-band token based policy enforcement, and Control Plane token based signalling in VLSR networks. TBN use case is described in details in section 6. In this case part of the TVS functions and components are hardware based but use the same API and messaging. The proposed token handling model allows for integration of the circuit provisioning and application flow provisioning as different layers of the token based enforcement model.

4.3.2 TVS integration into OSCARS/DRAGON infrastructure

Figure 4.9 below illustrates how TVS can be used in OSCARS/DRAGON network provisioning infrastructure [35, 36].

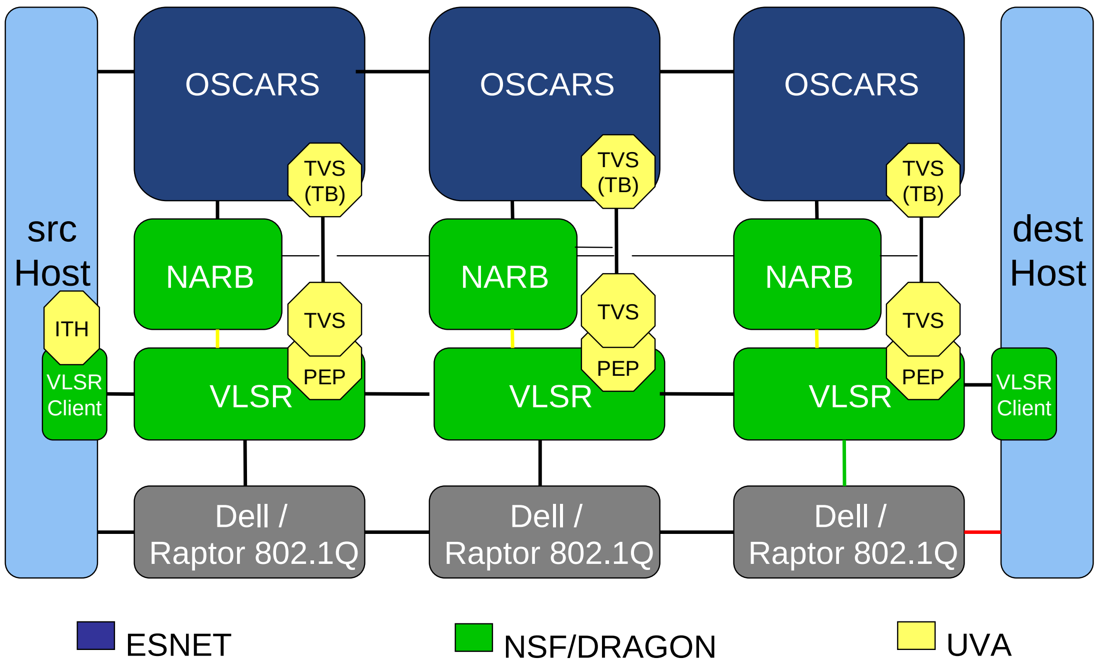

Fig. 4.9. Functional component overview OSCARS / DRAGON integration using GMPLS token based scenario (TVS – Token Validation Service; TB – Token Builder; ITH – Insert Token Here function; PEP – policy Enforcement Point)

The following shortly describes a simple resource reservation and token based access control in the multidomain OSCARS/DRAGON based network provisioning infrastructure (note, most of actual network related operational details are dropped to focus only on TVS related functions and operations):

1) Resource reservation is requested by the user (VLSR) client and initiated by the first domain which generates a Global Reservation ID (GRI).
2) The first domain (and each next domain) makes a reservation for required ingress/egress connections, store this information and may create also Local Reservation ID (LRI) that is linked to the GRI.
3) The first domain (and each successive domain) sends a reservation request to the next domain including required service/resource description and the GRI.
4) Each domain keeps GRI together with LRI until the whole path is confirmed or discarded.
5) When the reservation process reaches the last domain, the local reservation service needs to make a binding between global reservation process and a local reservation. This is done by generation a token and additionally a token key. Such a function is provided by the Token Builder (TB) of the TVS.
6) Two sets of information are stored in each domain:
    - GRI, LRI and related reservation information is stored by resource reservation/management service in each domain.

- "token – GRI – (LRI) – (token key)" is stored by TVS.

7) Token and optionally token key are returned to the user client. In a simple scenario, token will be included into following user requests or dataflow originated from the user. In more advanced scenario, a token will be generated by the user client based on token key.
8) When a domain (or InterDomain Controller (IDC)) receives a service request, it extracts the token calls a PEP (additionally providing also any necessary access control related information).
9) In a simple scenario, when receiving the token, PEP request the TVS to validate the token. TVS performs this by checking "token – GRI – (LRI) – (token key)" table stored by TVS. TVS replies with the Boolean "Yes/No" answer.
   - This is a responsibility of VLSR and OSCARS to provide a requested service if PEP-TVS response is positive.
10) Token validity may expire automatically or token related TVS table entry can be invalidated by IDC. In this case PEP-TVS will return negative answer.

It is important to mention that there is some background configuration and setup required to enable interdomain TVS operation. In the proposed TVS implementation such configuration ability is provided as a part of the general GAAAPI deployment and configuration.

4.3.3 TVS functional requirements

TVS architecture and design should support the following functionality:

1) Basic/simple TVS functionality includes checking validity of a token.

2) Extended TVS functionality should allow token re-building when sending dataflow to or requesting service from the next domain. (Note, this functionality should be supported by a sound dynamic secure association management model, simple solution for which is proposed below as token handling model.)

3) Additionally, TVS may be required to support token or token key distribution at the reservation stage or at the stage of the reserved resource deployment/configuration.

4) Token building function is an incumbent function of the TVS and consequently it is supported by the Token Builder (TB or TVS-TB) component. TB function should allow generating token key and token as derivative from the GRI. Additionally TB should allow generating token dynamically using token key and variable dataflow data, e.g. IP packets payload as in case of TBN.

5) TVS implementation should support both in-band dataflow token-based signalling and control plane signalling using XML-based tokens. As a consequence of intended use for in-band token-based signalling, token key and token should be of fixed length.

6) TVS should maintain own run-time table "token – GRI – (LRI) – (token key)", where LRI and token key are optional. Additionally The TVS table may contain a status or validity period of the tuple "token – GRI – (LRI) – (token key)". GRI and/or LRI will link to actual local resource reservation table maintained by the resource

reservation and management service and contain all necessary details. The TVS table should be stored in non-volatile memory to keep its state between start-up/initialisation periods.

7) TVS should allow integration into control plane or data plane using simple API. This should constitute a basic TVS functionality.

8) TVS should allow smooth integration into more general AAA infrastructure and support multidomain resource reservation/authorisation. In this case, TVS may be implemented as a component of more general multidomain complex resource authorisation infrastructure to support authorisation session management.

4.3.4 Token handling model

The token generation and handling model is based on the shared secret HMAC-SHA1 algorithm [37]. The TokenKey is generated in the following way:

TokenKey = HMAC(GRI, tb_secret)

where

GRI – global reservation identifier,
tb_secret – shared Token Builder secret.

A token is created in a similar way but using TokenKey as a HMAC secret:

TokenValue = HMAC(GRI, TokenKey)

This algorithm allows for chaining token generation and validation process, e.g.:

“GRI-TokenKey-TokenValue => LRI-l_TokenKey-l_Token”

The key management model is not discussed at this stage of the project. The token handling model relies on the shared secret that is installed at all participating NRPS nodes. It is being investigated that current model can be replaced with the IBC (Identity Based Cryptography) [38, 39] that will allow to replace shared secret token handling model that has know manageability problems.

The current TVS implementation allows handling two types of tokens in binary and XML format. In both cases reservation token is tuple of GRI and TokenKey that should be included into the request or service request.

5 GAAA-AuthZ implementation in the GAAA Toolkit

This section provides information about current implementation of the basic GAAA-AuthZ functionality as pluggable services and components of the GAAA Toolkit with reference to the existing code [40, 41].

5.1 Extended GAAA Toolkit Functionality to support dynamic services provisioning

The GAAA Toolkit provides two profiles for general Policy Based Access Control (GAAA-PBAC) and extended profile for Complex Resource Provisioning (GAAA-CRP). Suggested GAAA-CRP and GAAA-PBAC structure is shown on Figure 5.1 below. It contains the following main functional sub-systems:

- GAAA-PBAC subsystem that provides GAAA-PBAC profile functionality and basically includes PEP, PDP and GAAAPI with related Application Specific Modules (ASM);
- GAAA-CRP subsystem includes GAAA-PBAC subsystem used for general policy evaluation and adds flow control with the Flow Control Engine (FCE) and Flow Repository modules;
- Rule Based Engine (RBE) is represented by combination of PDP used for individual policies evaluation and FCE that control multiple policies evaluation or other sequence of policy evaluation for the complex resource;
- GAAAPI that provides all necessary functionality for the communication between PEP and PDP.
- A set of ASM's that provide interfaces to application specific functions of the requestor (requesting service) and the resource/service.

Technically, two defined GAAA profiles use the same set of functional components but have different configuration of modules/components related to security context (including key, trust relations, external call-outs configuration), internal components interaction and also required ASM functionality.

GAAAPI consists of a number of functional modules which functions don't fall under the generic PEP or PDP definition and provide security context transformation and handling when sending requests/responses between PEP and PDP, ensure in this way interoperability between possibly different PEP and PDP implementations. In particular case, GAAAPI includes:

- a Triage and Cache used to provide initial evaluation of the request including validity of provided credentials, when validating AuthZ tickets and tokens, the Triage can call to other modules:

- a Token Validation Service (TVS) to validate AuthZ or reservation token;
- A Ticket Authority that provide more complex functions for AuthZ ticket building and evaluation;

- a Namespace Resolver to define and resolve what policy and what attributes should be used for the request evaluation
- and Attribute Resolver and Policy Information Point (PIP) provide resolution and call-outs to related authoritative Policy Authority Points (PAP) and Attribute Authority Service (AAS) which can be a part of general Identity Provider service (IdP).

Tickets and tokens handling functionality allows for AuthZ session handling without requesting typically slow policy evaluation process.

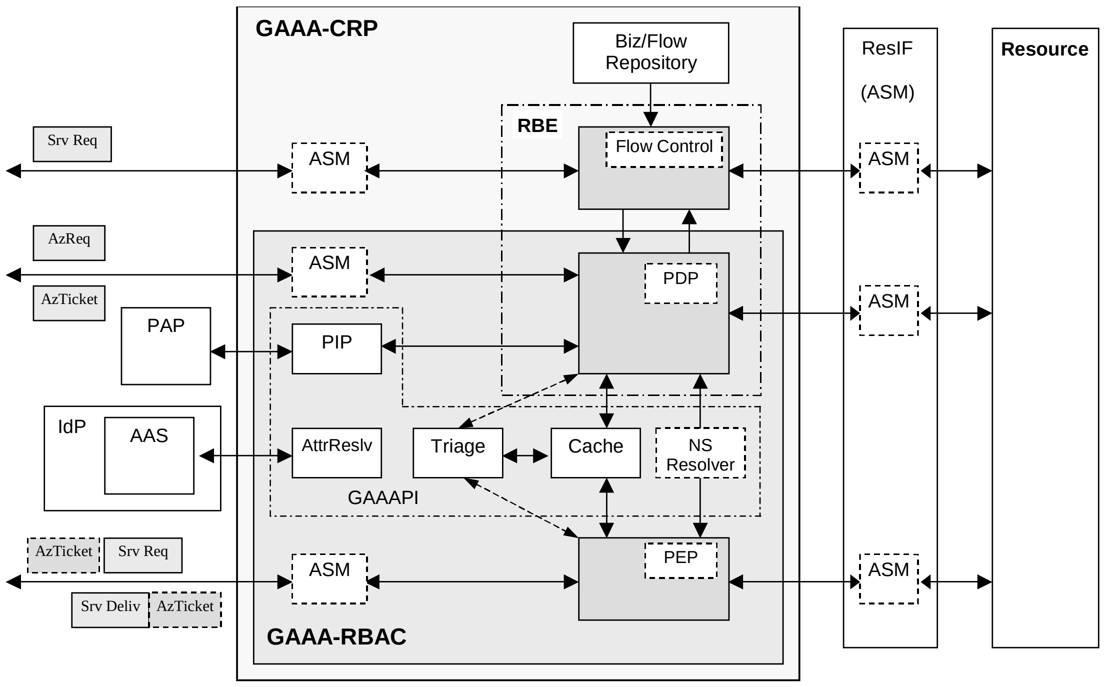

Figure 5.1. GAAA-CRP and GAAA-RBAC structure and main functional components

Separation of the flow management/processing and individual resources' policy evaluation will allow to separate business related part of the provisioning process that is normally related to the general/complex user request and policies applied to some component services/resources. Service/resource policies are more static and managed by owners/providers. Provider of the complex services/resources can apply it's own provisioning (business) model that can be described in the form of (work)flow and can contain different options for that provisioning and consequently different sequence of individual policies evaluation and also some other conditions related to overall provisioning process.

Workflow and (resource) policy separation doesn't affect individual policies evaluation that can also have some sequence of evaluation of the request against the related/referenced policy. In this relation there can be defined three levels/steps of the service request evaluation against the provisioning or individual policy:

- one step (or instant) request evaluation by Triage that simply checks (instant) request matching against provided AuthZ ticket/token or instant push-policy;
- resource/service policy evaluation by the PDP that does request evaluation according to the policy that itself describes a sequence of provided attributes/information evaluation, e.g. in XACML evaluation sequence includes first target (subject, resource, action) matching, next rules evaluation and finally rules combination to make overall policy based decision;
- complex request evaluation that requires multiple policies evaluation in the sequence described by provider or request specific (business) flow; in this case the FCE take care about driving the evaluation and provisioning process.

Outsourcing combination of individual policies evaluation to upper layer element/functionality of FCE will simplify multiple policies management in sense that there will not be a need for the overall policy validation to avoid possible conflicts and attributes conversion.

5.2 GAAAPI Implementation

5.2.1 GAAAPI functionality overview

GAAAPI package provides all necessary functionality to smoothly integrate AAA/AuthZ services into target application. GAAAPI package is provided together with the PEP implementation and simple XACML PDP implementation.

PEP-GAAAPI is called from the application AuthZ gateway that extract necessary information from the service request and creates AuthZ request to PEP. GAAAPI functionality supports all necessary communication between PEP and PDP and depending on implementation may include also external callout to such components as PDP, PAP, Attribute Authority Service (AAS), TVS, and Obligation Handlers (OH).

Current GAAAPI supports only local call to internal/local components. Further GAAAPI development will include external callouts to domain or site central AAA/AuthZ services.

5.2.2 PEP-GAAAPI interface

PEP-GAAAPI interface provides a few commands/methods to request policy based AuthZ decision depending on the set of provided information:

a) Method #1 should either return a logical value "True" or "False", or throw the appropriate exception

```
       Boolean authorizeAction (String resourceId, String actions, HashMap subjmap)
       throws java.lang.Exception,
       org.aaaarch.gaaapi.NotAuthenticatedException,
       /* user subjconfdata (i.e. authenticationToken) is not valid */
       org.aaaarch.gaaapi.NotAvailablePDPException;
       /* PEP could not reach PDP, or other internal error*/
where
@ resourceId – Resource ID
@ actions – requested actions (currently only one action)
@ {subject} set of values (subjectId, subjconfdata, role, subjctx)
@ subjectId - subject ID, can be null as it is not used in AuthZ
@ subjconfdata - AuthN token, can be null as it is not used in policy evaluation
@ role - role for the particular request
@ subjctx - subject context, e.g. VO, VLab or Experiment in which role is defined
```

b) Method #2 should either return a logical value "True" or "False", or throw the appropriate exception

```
        Boolean authorizeAction (String resourceId, String actions, String subjectId,
        String subjconfdata, String roles, String subjctx)
        throws java.lang.Exception,
        org.aaaarch.gaaapi.NotAuthenticatedException,
        /* user subjconfdata (i.e. authenticationToken) is not valid */
        org.aaaarch.gaaapi.NotAvailablePDPException;
        /* PEP could not reach PDP, or other internal error*/
```

c) Method #3 should either return a valid AuthorisationTicket (or AuthorisationToken), or throw the appropriate exception

```
        String authorizeAction(String authzTicketToken, String sessionId, String
        resourceId, String actions)
        throws java.lang.Exception,
        org.aaaarch.gaaapi.NotAuthenticatedException,
        /* user subjconfdata (i.e. authenticationToken) is not valid */
        org.aaaarch.gaaapi.NotAuthorizedException,
        /* user is not allowed to perform requested action */
        org.aaaarch.gaaapi.NotAvailablePDPException;
        /* PEP could not reach PDP, or other internal error*/
were
@ authzTicketToken – AuthZ ticket or token containing all necessary AuthZ session context
@ sessionId – Session ID that can be also a Global or Local reservation ID (LRI/GRI)
```

d) Method #4 should either return a valid AuthorisationTicket (or AuthorisationToken), or throw the appropriate exception

```
        String authorizeAction (String authzTicketToken, String sessionId, String
        resourceId, String actions, HashMap subjmap)
        throws java.lang.Exception,
        org.aaaarch.gaaapi.NotAuthenticatedException,
        /* user subjconfdata (i.e. authenticationToken) is not valid */
        org.aaaarch.gaaapi.NotAuthorizedException,
        /* user is not allowed to perform requested action */
        org.aaaarch.gaaapi.NotAvailablePDPException;
        /* PEP could not reach PDP, or other internal error*/
```

5.2.3 Authorisation ticket and token format

GAAAPI supports AuthZ and AuthN tickets generation in a proprietary XML format and by using the SAML assertion format, which format was discussion in section 4.2.2. Examples of AuthZ tickets are provided in the Appendix B.

5.3 TVS implementation

5.3.1 TVS version 0.1 Overview

TVS version 0.1 is implemented as a part of the general GAAAPI Java package and provides required functionality that can be integrated into the target network provisioning systems and applications, in particular OSCARS and DRAGON.

All basic TVS/TB functions are accessible and requested via Java API. This is required to make TVS reaction as quick as possible. Some off-band TVS functions can be operated via WS/SOAP based interface as for example the TVS table programming.

Further TVS development will extend WS interface to allow all TVS functions be accessible via Web services. This should allow creation of the TVS infrastructure for interdomain key distribution and tokens management.

5.3.2 TVS interface

a) Token Builder commands

```
GetToken (GRI, TokenKey?)
```

b) PEP-TVS interface

Boolean ValidateToken (TokenValue, GRI?, TokenKey?)
Boolean ValidateXMLToken (XMLToken, TokenKey?)

c) PEP Interface for using with Token based credentials/enforcement

This interface is a part of the general GAAAPI package functionality. It uses the same commands and methods as GAAAPI PEP interface.

d) Internal TVS programming interface contains two basic commands:

SetEntryTVStable (GRI, (TokenValue | TokenKey)?, NotBefore?, NotOnOrAfter?)
DeleteEntryTVStable (GRI, LRI?) –

e) External/WS interface

External TVS interface will allow programming TVS table by sending particular reservation information. The message format may contain the following information:

MessageSetTVS (GRI, ResourceID, TokenValue, LRI?, TokenKey?, NotBefore, NotOnOrAfter)

Note: In current implementation this information is expressed and communicated in a form of the AuthZ ticket.

5.3.3 XML Token format

XML token format uses a special “TVS/TBN” profile of the more general AuthzToken format. Example of the full TVS XML token is shown below:

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
Issuer="urn:aaa:gaaapi:token:TVS" SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
TokenId="d1384ab54bd464d95549ee65cb172eb7">
    AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
    AAA:Conditions NotBefore="2007-08-12T16:00:29.593Z" NotOnOrAfter="2007-08-
13T16:00:29.593Z"/>
</AAA:AuthzToken>
```

where the element `<TokenValue>` and attributes `SessionId` and `TokenId` are mandatory, and the element `<Conditions>` and attributes `Issuer, NotBefore, NotOnOrAfter` are optional.

Minimum token format is illustrated in the following example:

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
TokenId="d1384ab54bd464d95549ee65cb172eb7">
    AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
</AAA:AuthzToken>
```

Attributes “Issuer” allow for distinguishing different AuthzToken profiles. The TVS profile is identified by the URN "urn:aaa:gaaapi:token:TVS".

5.3.4 TVS package

TVS package currently is implemented as a part of the general GAAAPI package to share common configuration and security profiles.

TVS related classes are organised as a **org.aaaarch.gaaapi.tvs** package. All interfaces are supported by corresponding method of the TVS.java class.

Token builder functions can be accessed either via TVS.java or directly from TokenBuilder.java class

org.aaaarch.gaaapi.tvs package relies on some other supporting class in the core org.aaaarch.gaaapi package but number of external to the **org.aaaarch.gaaapi.tvs** package classes is limited and at later versions the TVS package will be provided as a self-consistent package.

The package is provided with the TestTVS.java test class that allows for interactive test of all available functions.

In current implementation the TokenBuilder secret tb_secret is the secret phrase 3DES encrypted and hard coded into the Token Builder classes. Following TVS releases will provide secure TB configuration that will be bound to the domain’s or site’s public key.

The package requires a special supporting directories structure for storing keys, schemas and temporal files.

6 Token Based Networking with in-band policy enforcement architecture and implementation

The section provides an overview of the Token-Based Networking (TBN) and then describes the token-based policy enforcement mechanism used in the TBN implementation. TBN is being developed by UvA as a solution of adding in-band policy enforcement function to on-demand network resources provisioning [44 – 45].

6.1 Overview

The TBN architecture uses the push model of a generic AAA framework. The push model of authorisation is compatible with either form of signalling. In the token based switch over IP (TBS-IP) we opt for in-band signalling for reasons of flexibility resulting from the per-packet granularity. Specifically, we insert tokens into each IP packet as proof of authorisation. Tokens are a simple way to authorise resource usage which may convey different semantics. For instance, we may specify that only packets with the appropriate token are allowed to use a pre-established network connection in a specific time-frame and embed these tokens in the packets of an application distributed over many IP addresses. [44]. An advantage of the push model is that time of authorisation is decoupled from time of use.

Token Based Switch (TBS) is a low-level system for traffic routing at high speeds (multi gigabits/sec) based on packet authentication. TBS helps high-performance computing and grid applications that require high bandwidth links between grid nodes to bypass the regular Internet for authorised packets by establishing shortcuts on network links with policy constraints.

TBS is fast and safe and uses the latest network processor generation (Intel IXP2850). TBS is feasible at multigigabit link rates. In addition, it has the following goals: (1) path selection with in-band admission control (specific tokens gives access to shortcut links), (2) external control for negotiating access conditions (e.g., to determine which tokens give access to which links), and (3) secured access control.

Figure 6.1 and Figure 6.2 illustrate a few use cases for a TBS-IP router usage in multi-domain networks.

A TBS-IP system works as follows: the client (e.g., user/application requestor in Figure 6.1) contacts an AAA server separate from the datapath to obtain authorisation for the use of network resources (for instance, the optical shortcut B). Then, the AAA server checks the credit of the client and also the availability of the requested network resources. When both conditions are positive, the AAA server generates an authorisation ticket (AuthZ ticket) that contains as proof of authorisation the following items: a unique Global Resource Identifier (GRI), a key (TokenKey), and a description of the required lightpath across the network. The AAA server sends the AuthZ ticket to every policy enforcement point (PEP) along the required path and also back to the client. Next, when the client is ready to use the resources, the client pushes the proof of such authorisation to the service equipment such as the Token Validation Service in a token builder module (TB/TVS) on the network device. TB/TVS module checks the validity of the AuthZ ticket and generates a proper token for each data packet that goes out of the client's system. This token travels together with data across the networks. Every PEP across a multi-domain network (TBS-IP domain1, TBS-IP domain2, etc.) has specific hardware that checks the built-in token from each received data packet against a local AuthZ ticket. When the PEP's check is positive, then data packet takes an authorised path. Otherwise, it takes a default path such as slow routed networks. In other words, the proof of authorisation (token) drives the data over the networks in order to reach the destination within the required specifications (bandwidth, delay, etc.) by making use of specific network resources such as lightpath shortcuts (e.g., route (B) in Figure 6.1).

Figure 6.2 shows another context where TBS-IP systems work together with TBS-GMPLS. In other words, in this scheme we suppose to have some GMPLS lightpaths within the end-to-end connection that uses normally IP packets.

A Token Based Switch (TBS) is hardware and software system that receives authorisation requests for certain IP packet flows from a high-level authority (AAA servers) and allows for intensive packet processing (encryption operations) at high speeds (multigigabits/sec). This system is going to be plugged into an AAA framework for high speeds lightpaths selections especially when the traffic crosses multi-domain networks. In order to achieve the above mentioned requirements, we need to design and develop a modular system that separate the control path by data path in a manner that also provides standard and secured communications to different levels of components. In this respect, we use SOAP/XML for high-level and ForCES for the low-level communications.

Summarising, each TBS-IP system plugged in multi-domain networks (as shown in Figure 6.1) provides packet routing based on a global resource identifier (GRI). This GRI is a value unique per end-to-end lightpath and eventually issued per application (not a host, as an application might use multiple physical hosts).

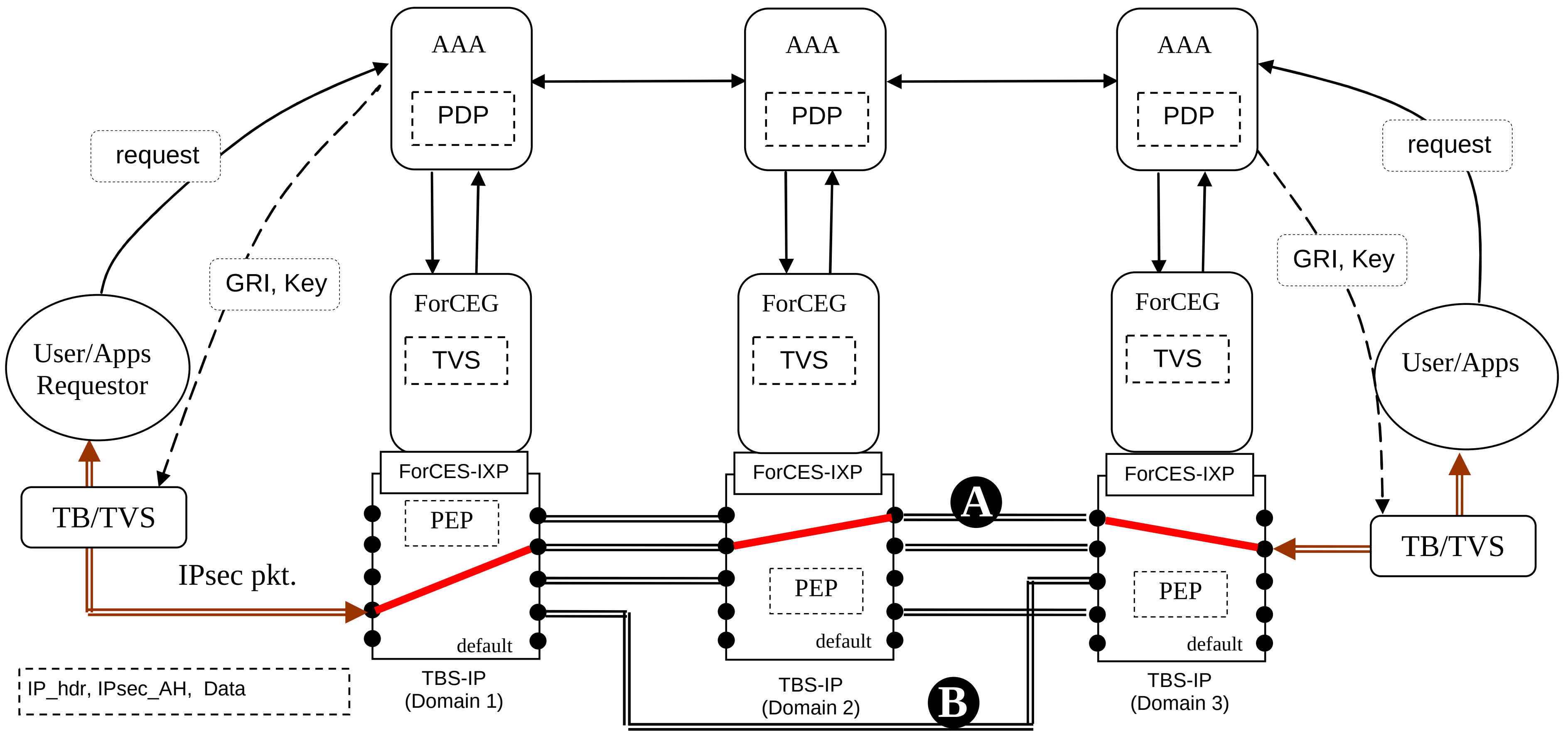

Figure 6.1. Providing end-to-end lightpaths over multidomain networks using TBS-IP systems interconnected into an AAA framework.

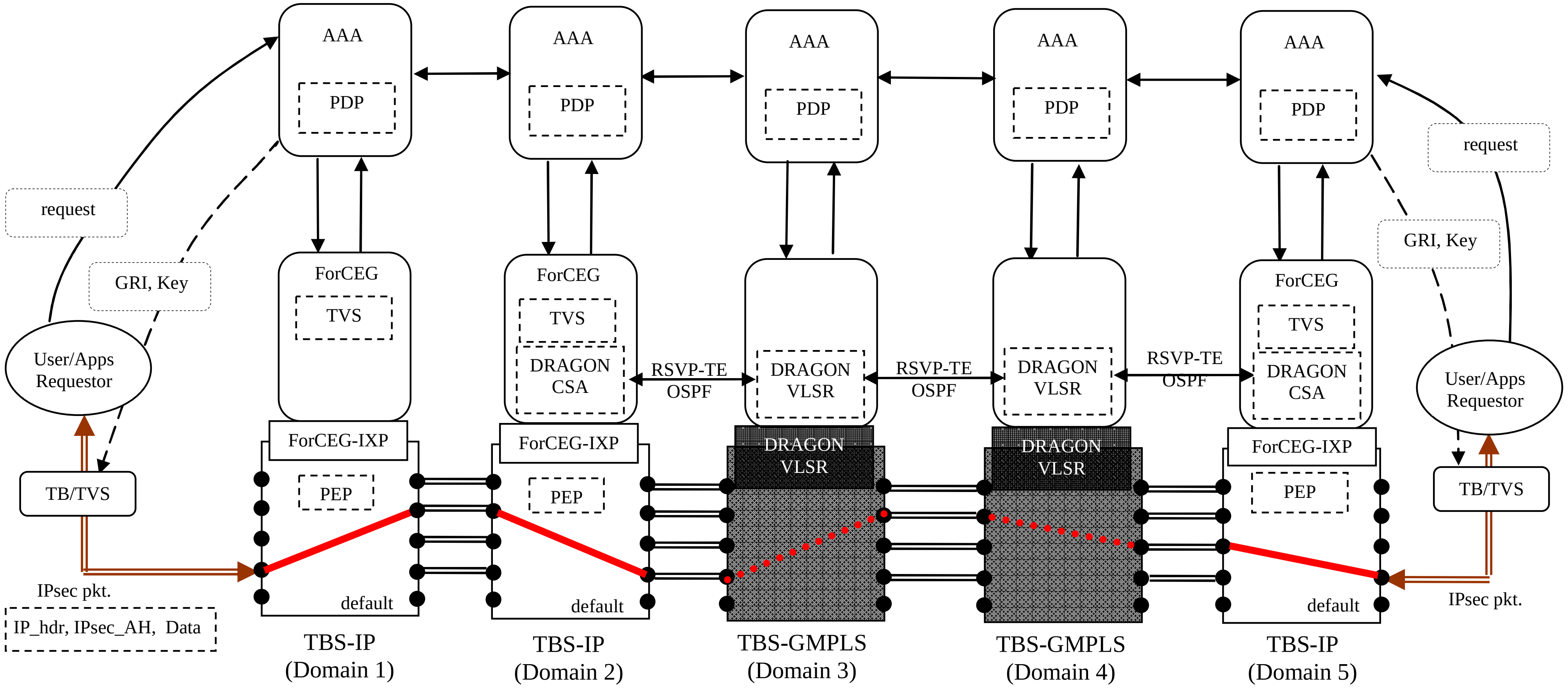

Figure 6.2. Providing end-to-end lightpaths over multidomain networks using a mix of TBS-IP with GMPLS systems.

6.1.1 TBS-IP development

The TBS architecture offers to the user applications an authenticated access control mechanism to critical high-speed links (lightpaths) across multi-domain hybrid networks. The procedure consists of two phases that are decoupled in time: (1) a high-level set-up phase (obtaining tokens from an AAA web-service), and (2) a fast datapath consisting of low-level authorisation checks (per-packet token checks at network edges within a multi-domain end-to-end connection). In other words, the first phase allows individual users or group of users (e.g., a research institution), or even user applications, to request privileged end-to-end connection across multi-domain networks by contacting only one authority: their own ISP. The second phase determines how TBS authenticates network traffic (TCP connections, UDP transmissions, or other protocols on top of IP protocol) and how it checks the traffic for authorisation on behalf of their applications. The second phase is also responsible for preventing malicious use of lightpaths in a multi-domain network. From practical point of view, two network components are involved in the datapath: the token builder (TB) and the token switch (TS).

We identified the following goals:

- Implement the token over IP principles inside the data path software modules, token builder (TB) and token switch (TS), developed on a network processor hardware architecture. The software modules are remote configurable through ForCES standard interface that stands for FORwarding and Control Element Separation;

- The implementation builds modularly in order to have a single TBS-IP system that works on multiple application scenarios and hence, different low-level requirements, chosen at loading time through the ForCES control path. For example, TBS-IP/TB is a Token Builder configuration, TBS-IP/TS is a Token Switch configuration, or TBS-IP/TS-GMPLS is a Token Switch with gateway to GMPLS domains;

- The system provides interconnection to 3rd party systems such as Dragon-VLSR used in GMPLS and therefore, it provides token features over the GMPLS paths (TBS-IP/TS-GMPLS);

- The system is fully configurable in a safe manner by means of using secured Webservice as a high-level interface to an authority (AAA servers).

Summarising, the request is to create a software system that is modular, configurable for different application scenarios, and using standard communications and special hardware for packet processing at high speeds (network processors).

In the next section, Figure 6.3 shows schematically one Token Based Switch system at IP layer (TBS-IP) that uses a network processor hardware platform (IXP2850 dual-NPU). An IXP2850 network processor has one general purpose CPU (XScale) for control of the entire architecture composed of parallel specialised cores for traffic processing called microengines (μEngines).

6.2 TBS-IP architecture design

As shown in Figure 6.3, a host PC runs a webservice for the outside world interface (AAA server). The host PC connects to the specialised hardware for packet processing, IXDP2850, through a standard interface: ForCES [42]. The IXDP2850's control core (XScale) runs embedded linux that supports the control path of the ForCES interface, and each μEngine runs a custom packet processing task that implements specific forwarding elements like packet receiver, token builder, token switch, packet transmitter.

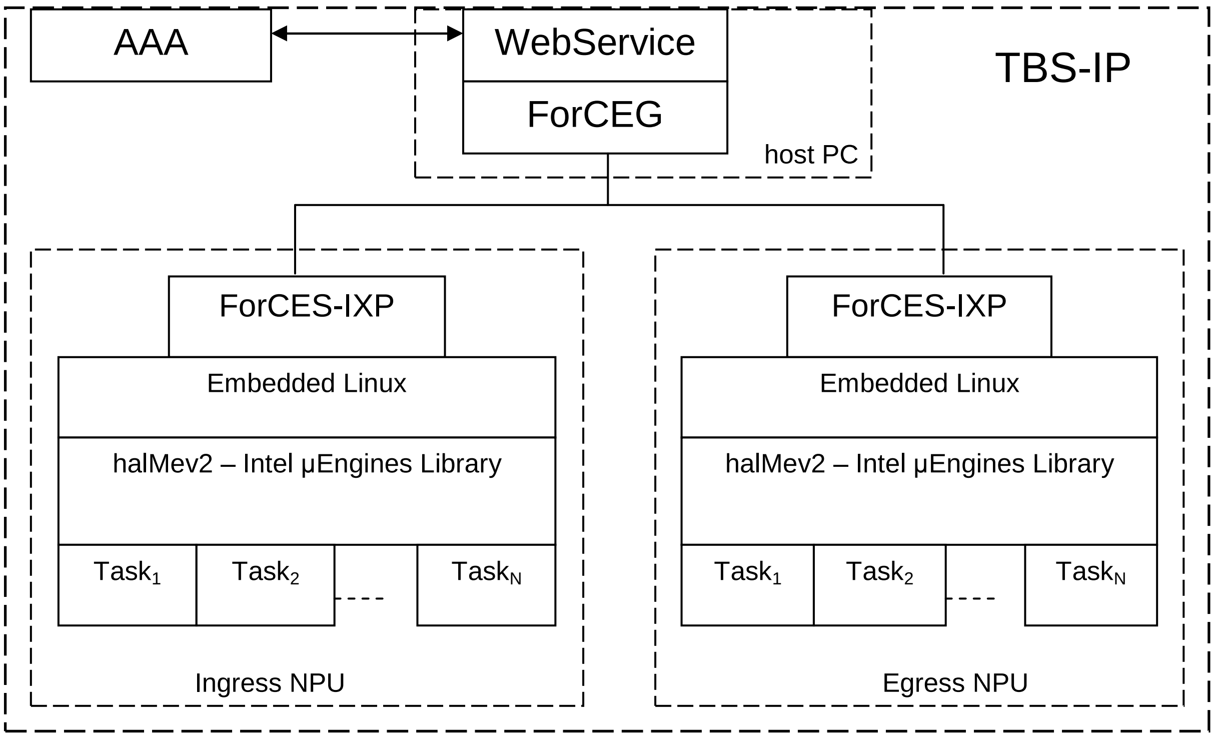

Figure 6.3. TBS-IP software architecture.

6.2.1 Requirements

Designing a Token Based Switch architecture following the ForCES standard guidelines we identified the following requirements:

- Modularity – provides flexibility in adding new features;
- Safety – prevents a rogue user to use privileged resources such as ligtpaths;
- Configurable – allows for various test-bed scenarios;
- Hiding low-level implementation details;

- Uses hardware specifically designed for network processing at high speeds (Network Processors) and also benefits of hardware supported encryption.

The TBS system has the followings states:

1. Self-Initialisation: hardware components (NPUs, memory, registers, etc.) and software components (O.S. loading, drivers loading, description tables, buffers);
2. A host connects to the TBS system remotely;
3. The host browses for the capabilities (features) within the current TBS system;
4. The host configures the connected TBS in a required state (e.g., Token Builder, Token Switch);
5. The host starts the system;
6. The host updates in run-time the required parameters in the TBS system: adds new authorised TokenKeys, removes 'expired' TokenKeys from the AuthorisationTable;
7. The host fetches periodically some of the debugging information (e.g., authorised packets, unauthorised packets);

6.2.2 Software item architectural design

This section describes what software modules are needed and how they are used.

As illustrated in Figure 6.4, the system is composed of the following software modules, described in a bottom-up approach (low-level, data-path, to high-level, control path):

FIX2850, having the following sub-modules:
  Rx;
  Tx;
  TB (TokenBuilder);
  TS (TokenSwitch);
ForCES-IXP, having the following sub-modules:
  Init_HW, Init_SW, ueManager;
  FE-IXP server (over TCP);
GMPLS gateway;
ForCEG-WebService interface to the outside-world.

6.2.3 Interface design

Figure 6.4 also highlights the interfaces used in the TBS-IP system:

1) outside world via WebServices;
2) CE-CE using a simple communication over TCP;
3) CE-FE using ForCES standard over TCP;
4) FE-LFBs (control – data path specific implementation for IXP network processors).

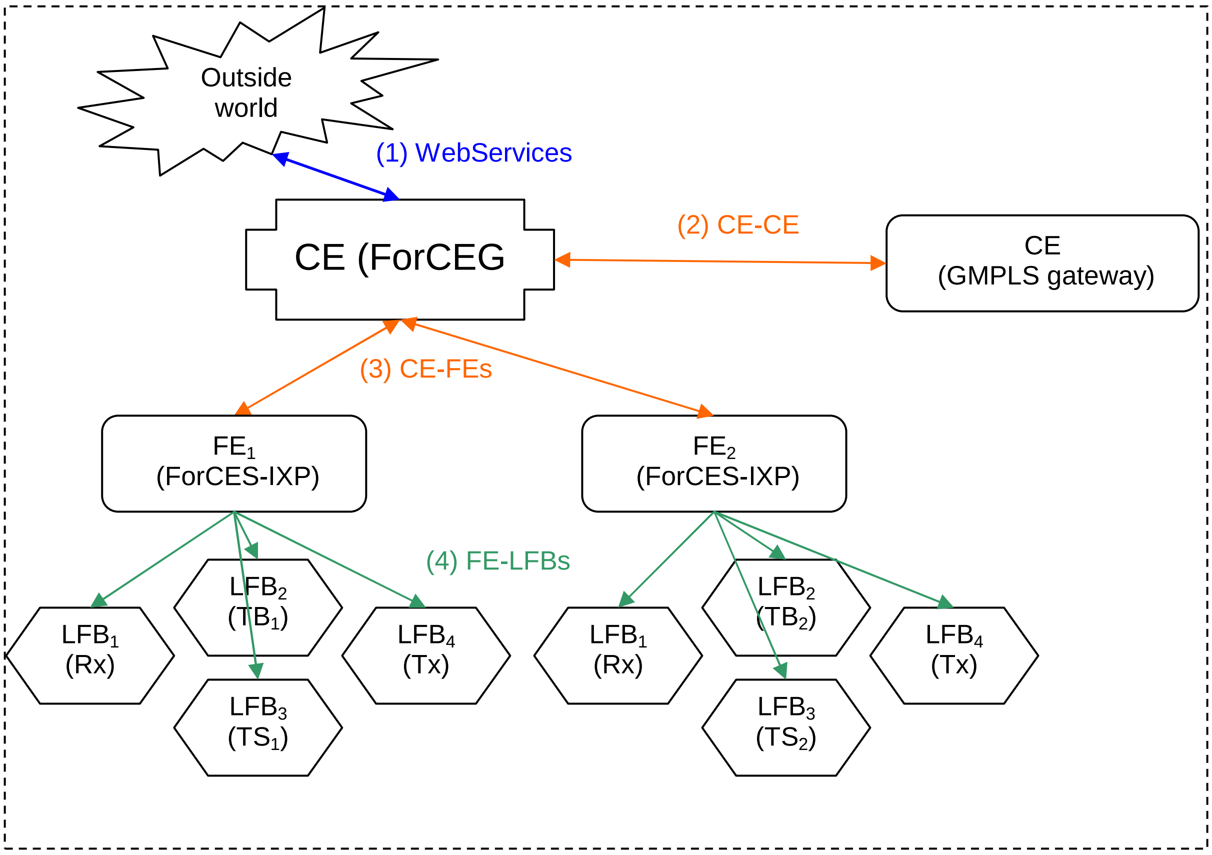

Figure 6.4. Interfaces used in the TBS-IP system

Although Figure 6.4 shows all the interfaces the TBS-IP system might use, we identify practically only the following four configuration schemes for different usage of TBS-IP:

1) TBS-IP/TB (Token Builder);
2) TBS-IP/TS (Token Switch);
3) TBS-IP/TS-TB (Token Switch and Builder);
4) TBS-IP/TS-GMPLS (Token Switch Gateway to GMPLS);

Each configuration is described simultaneously in the following sections on both levels: control and data paths.

6.2.4 Interface specification: TBS-IP/TB

TBS-IP/TB illustrates the Token Builder application. It is usually located at both ends of an end-to-end light-path. In other words, each user needs to have such a system that annotates the packets of authorised user applications with “tokens”.

In our dual-NPU implementation, we share the effective packet processing task for building and inserting the token (TB) between those two NPUs in order to benefit of all hardware encryption units available (2 units per NPU). The system works as follows: the received packets are stored into a shared packet buffer and made available for the next processing tasks (TB_1 and Tx) so as to load balance between these two tasks as shown in Figure 6.5, (1). One half of the received packets is further processed by the TB_1 task (2) and the other half is simple forwarded out to the second NPU (egress) by the Tx module (3). The Egress receiver does the same job as the one in Ingress, except that the load-balance is done by checking whether the packet has been already processed by the TB_1 in the first NPU or not. The processed packets are forwarded out of the system via Tx module, and the other packets are en-queued to the TB_2 for processing.

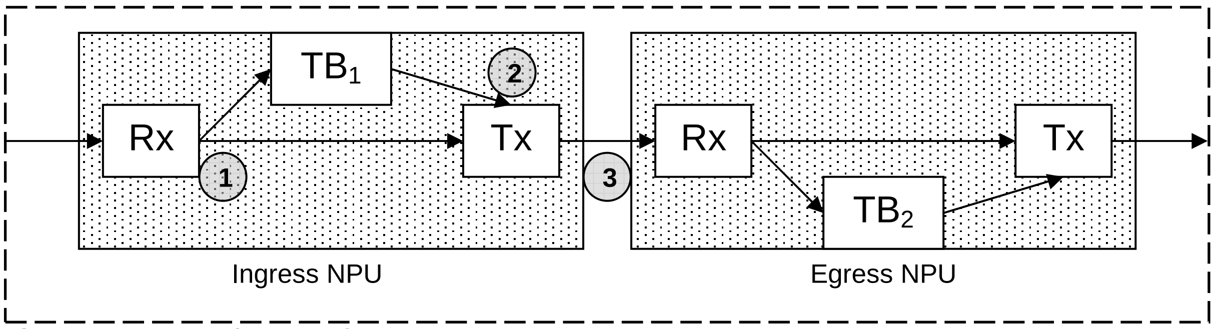

Figure 6.5. TBS-IP/TB Interface

The control path of this TBS-IP/TB application is shown in Figure 6.6. It illustrates the ForCES connections from CE (ForCEG running on a hostPC) to those two FEs (FE-IXP on Ingress and Egress NPUs).

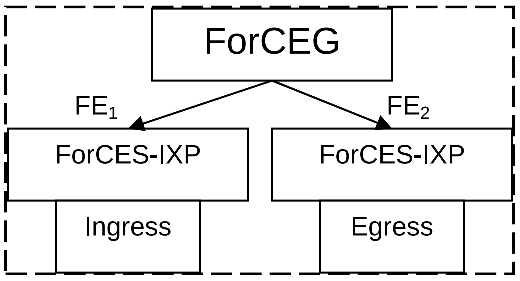

Figure 6.6. ForCES connections from CE (ForCEG) to two FEs

6.2.5 Interface specification: TBS-IP/TS

TBS-IP/TS describes the Token Switch application and is illustrated in Figure 6.7. TBS-IP/TS is located at every domain border across a multi-domain light-path. This application decides which packets should take a certain 'short-cut' (part of the established light-path) or should be forwarded out to the routed network (Internet).

Same as in TBS-IP/TB application, we need to make use of the dual-NPU features and hence, we share the effective packet processing task for checking the token (TS) between those two NPUs.

The control path of TBS-IP/TS application is same as the one used in TBS-IP/TB as shown in Figure 6.6.

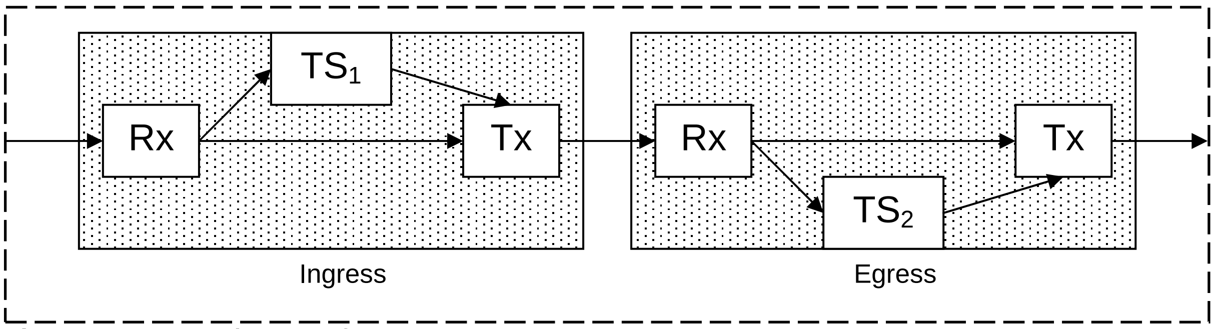

Figure 6.7. TBS-IP/TS Interface

6.2.6 Interface specification: TBS-IP/TS-TB

TBS-IP/TS-TB describes a combination of the above mentioned two applications: Token Builder and Token Switch. It can be located in every domain border across a multi-domain light-path by replacing the simple Token Switch application when the next domain requires a different key/identifier than the current domain. This application, like the simple Token Switch, decides which packets should take a certain ‘short-cut’ (part of the established light-path) or should be forwarded out to the routed network (Internet). In addition to the TokenSwitch, the TBS-IP/TS-TB application also rebuilds the token of every outgoing packet to the authorised ports.

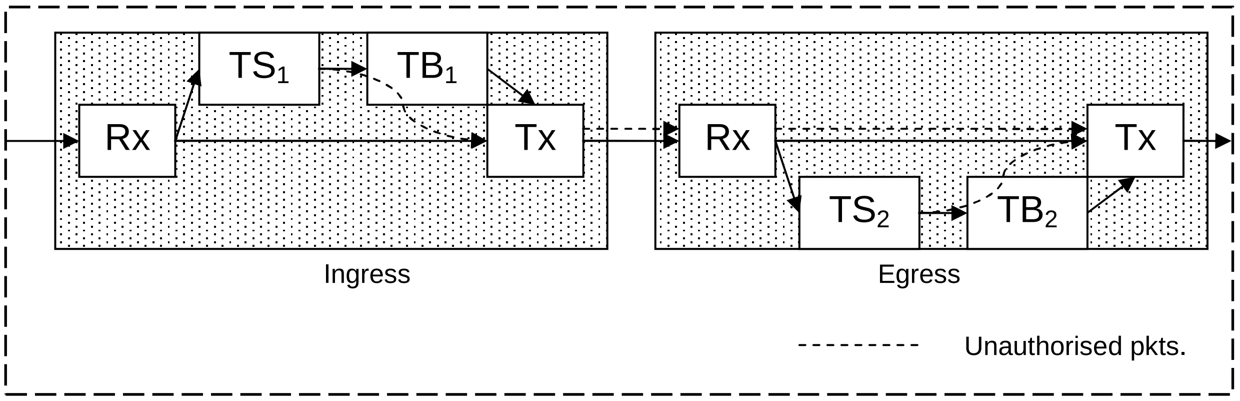

Figure 6.8. TBS-IP/TS-TB Interface

Same as in TBS-IP/TB application, we make use of the dual-NPU features and hence, we share the effective packet processing task for checking the token (TS) and for building a new token between those two NPUs. We chosen to map the TS, TB modules onto those two NPUs as shown in Figure 6.8 because of the following facts gained from a demo system built previously: the most computation required by the TS module consists of encryption, while TB also uses lots of memory operations for token insertion in the IP packet. Therefore, the chosen mapping (TS-TB pairs on each NPU) fits better than an eventually TS-TS / TB-TB mapping.

The control path of TBS-IP/TS-TB application is same as the one used in the previous two applications as shown in Figure 6.6.

6.2.7 Interface specification: TBS-IP/TS-GMPLS

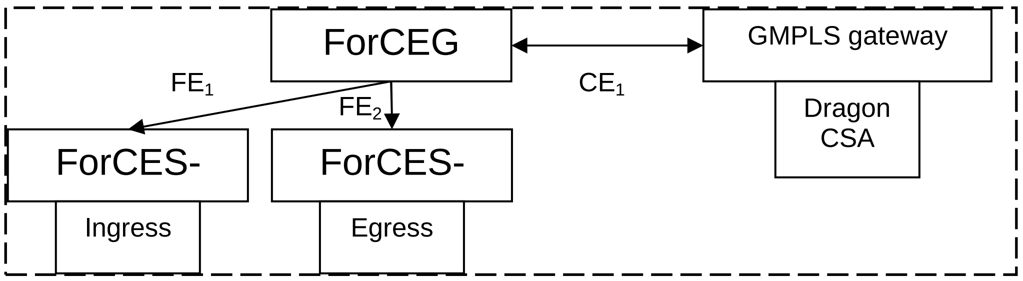

Figure 6.9. TBS-IP/TS-GMPLS Interface.

TBS-IP/TS-GMPLS is another application that has special requirements for routed packets across two domains that use different technologies (one uses tokens over IP, while the other one might use tokens over GMPLS). In this case, we use for the data-path the same mapping as in TBS-IP/TS application, but for control path we need to connect to the control path of the GMPLS system that is called Dragon-CSA framework. Therefore, as shown in Figure 6.9, the ForCES architecture uses one CE interface that consists of the connection to the CSA system.

6.2.8 Interface specification: TBS-IP to outside world (AAA server)

Figure 6.10 shows the interface between the outside world (AAA server) and the ForCEG.

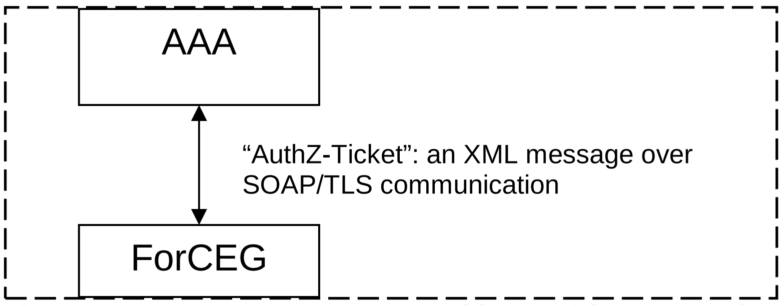

Figure 6.10. Interface TBS-IP to outside world (AAA server)

6.3 ForCES protocol in a TBS-IP router

The ForCES protocol [40] works in a master-slave mode in which FEs are slaves and CEs are masters. Information exchanged between FEs and CEs makes extensive use of packets in the type of Type-Length-Value (TLV). The protocol includes commands for transport of LFB configuration information, association setup, status and event notifications, etc.

The ForCES Protocol is only used for transporting commands between the Control Plane Elements and the Forwarding Elements. All ForCES Protocol packets have a common header and the rest of the body differs depending on the ForCES model and the part of the FE, the CE will send a message to.

6.3.1 ForCES Model

The FE model is based on an abstraction of distinct logical functional blocks (LFBs), which are interconnected in a directed graph, and receive, process, modify, and transmit packets along with metadata [42, 43]. The FE model is designed such that different implementations of the forwarding datapath can be logically mapped onto the model with the functionality and sequence of operations correctly captured. However, the model is not intended to directly address how a particular implementation maps to an LFB topology. It is left to the forwarding plane vendors to define how the FE functionality is represented using the FE model. The goal of the IETF ForCES Model group, is to design the FE model such that it is flexible enough to accommodate most common implementations.

The LFB topology model for a particular datapath implementation must correctly capture the sequence of operations on the packet. The ForCES base protocol is used by the CEs and FEs to maintain the communication channel between the CEs and FEs. The ForCES protocol may be used to query and discover the inter-FE topology. The details of a particular datapath implementation inside an FE, including the LFB topology, along with the operational capabilities and attributes of each individual LFB, are conveyed to the CE within information elements in the ForCES protocol. The model of an LFB class should define all of the information that needs to be exchanged between an FE and a CE for the proper configuration and management of that LFB.

Specifying the various payloads of the ForCES messages in a systematic fashion is difficult without a formal definition of the objects being configured and managed (the FE and the LFBs within). The FE Model document defines a set of classes and attributes for describing and manipulating the state of the LFBs within an FE. These class definitions themselves will generally not appear in the ForCES protocol. Rather, ForCES protocol operations will reference classes defined in this model, including relevant attributes and the defined operations.

Even though not absolutely required, it is beneficial to use a formal data modelling language to represent the conceptual FE model described in this document. Use of a formal language can help to enforce consistency and logical compatibility among LFBs. A full specification will be written using such a data modelling language. The formal definition of the LFB classes may facilitate the eventual automation of some of the code generation process and the functional validation of arbitrary LFB topologies. These class definitions form the LFB Library. Documents which describe LFB Classes are therefore referred to as LFB Library documents.

XML was chosen as the specification language in this document, because XML has the advantage of being both human and machine readable with widely available tools support [41].

6.3.2 Functionalities from ForCES required for CE-FE interface in TBS-IP

We model the interface between Control Element (CE) and multiple Forwarding Elements (FEs) using ForCES standard, as follows:

- **Association phase:**
  - CE interconnects to FE(s);
  - CE asks for capabilities to the FE(s);
  - FE(s) answers: (e.g., TS, TB);
  - CE requests to one FE to become TB, etc.;
  - FE then instructs HW to map the TB application;
- **Configuration phase:**
  - Add/Remove tuple, Changing in LFB attributes (e.g., Connector changes);
- **Events:** diagnostic messages from low-level reported to higher level;
- **Query:** reading of specific attributes of LFBs such as packet counters: pkts_count, processed_pkts_count

6.3.3 Modeling of applications in ForCES

We model the required function blocks for TBS-IP applications using ForCES standard, as follows:

- **FEs:** ingress, egress;
- **LFBs:** Rx, Tx, TB, TS, Tx_CSIX, Rx_CSIX;
- **Attributes:**
  - TUPLE (GRI, TokenKey, GRI_NextDomain, TokenKey_NextDomain, Port1, Port2, IPPacketMask, IPSource, IPDestination, PortSource, PortDestination, Status);
  - LOCATION (e.g., LFB.Rx located on hw_core 0);
  - Connector(val,val);
  - Pkt_count that is available in LFB(Rx, Tx, Tx_CSIX, Rx_CSIX);
  - Processed_pkt_count available in LFBs(TS, TB);

6.3.4 XML Model of TBS based on ForCES Model

Based on the ForCES Model draft which provides an XML schema for creating LFBs, there are only for distinct LFBs (Rx, Tx, TB and TS) because the Rx and Rx_CSIX have the same model, as Tx and Tx_CSIX. The actual XML can be found in **Appendix E**.

Each LFB has a Packet Processed counter and TS has an additional counter Bad Token Counter for each Tuple. The ForCEG will be able to register for an event based on the Bad Token Counter. If the Bad Token Counter exceeds a specific threshold, which will be set by the ForCEG, then the TS will send a message containing the GRI and the number of Bad Tokens.

6.4 Software item detailed design

The following software modules are described in this section:

- ForCEG high-level interface to the outside-world;
- ForCEG – plugins:
    - Message-parser;
    - TVS (Token Validation Service);
    - Gmp (GMPLS gateway);
    - APM (Advanced Power Management);
- FIX2850, having the following sub-modules:
    - Rx_Gig;
    - Tx_CSIX;
    - Rx_CSIX;
    - Tx_Gig;
    - TB (TokenBuilder);
    - TS (TokenSwitch);
- FE-IXP, having the following sub-modules:
    - FE (ForCES FE module implemented on IXP);
    - Init_HW, Init_SW, ueManager;
    - FE-IXP server (over TCP).

6.4.1 ForCEG high-level interface to the outside world

Figure 6.11 illustrates an example of Authorisation ticket AuthZ ticket as expressed in XML format.
Note that some of the AuthzTicket fields are not mandatory and hence, we provide specific mechanisms to derive local meanings from a global identifier. For instance, if AuthZ ticket has no specific field as "***TokenKey***", we chose to derive the TokenKey from a mandatory field ***SessionID*** also known as a global resource identifier (GRI). Thus, TokenKey = Encryption(GRI, secret), where the secret is a shared TVS specific phrase. However, GRI could be much bigger than that of the encryption algorithm required. For instance, using HMAC-SHA1 encryption algorithm, we use for deriving of the TokenKey only the first 20Bytes of the GRI.

```
<AuthzTicket xmlns="http://www.aaauthreach.org/ns/#AAA"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.aaauthreach.org/ns/#AAA
E:\Projects\Phosphorus\Workspace\AuthZ\azticket-ehalep.xsd" Issuer="urn:ph:trusted:tickauth:pdp"
TicketID="cba06d1a9df148cf4200ef8f3e4fd2b3" SessionID="00112233445566778899AABBCCDDEEFF00">
  <Decisions><Decision ResourceID=" http://resources.test-bed.phosphorus.eu/ForCES1"
result="Permit"/>
  </Decisions>
  <Resources>
      <Resource ResourceID="http://resources.test-bed.phosphorus.eu/ForCES1">
         <LRI purpose="MyDomain">00112233445566778899AABBCCDDEEFF00</LRI>
         <Key>AAAABBBBCCCCDDDDEEEE</Key>
         <Ports>
             <Port1>6</Port1>
             <Port2>8</Port2>
         </Ports>
         <ApplicationFlow>
               <IPpacketMask>A1A2A3A4A5A6A7A8A9B1B2B3B4B5B6B7B8B9C1C2</IPpacketMask>
               <IPsource>192.168.1.10</IPsource>
               <IPdestination>"192.168.1.100"</IPdestination>
               <PortSource>123456</PortSource>
               <PortDestination>654321</PortDestination>
         </ApplicationFlow>
      </Resource>
      <Resource ResourceID="http://resources.test-bed.phosphorus.eu/ForCES2">
         <LRI purpose="NextDomain">AA112233445566778899AABBCCDDEEFF00</LRI>
         <Key>AAAABBBBCCCCDDDDEEEE</Key>
         <Ports>
             <Port1>6</Port1>
             <Port2>8</Port2>
         </Ports>
         <ApplicationFlow>
             <IPpacketMask>A1A2A3A4A5A6A7A8A9B1B2B3B4B5B6B7B8B9C1C2</IPpacketMask>
             <IPsource>192.168.1.10</IPsource>
             <IPdestination>"192.168.1.100"</IPdestination>
             <PortSource>123456</PortSource>
             <PortDestination>654321</PortDestination>
         </ApplicationFlow>
      </Resource>
   </Resources>
   <Conditions NotBefore="2006-06-08T12:59:29.912Z" NotOnOrAfter="2006-06-09T12:59:29.912Z">
</Conditions>
</AuthzTicket>
```

Figure 6.11. Example of XML based AuthZ ticket format.

As also illustrated in Figure 6.12.[1], the Authorisation ticket is an XML message received by the ForCEG-Webservice module through a local WebService container (e.g., Apache Tomcat or Sun Application Server JWSDP). Next, this message is passed to the ForCEG software package. ForCEG will send the XML message to the “Message-parser” module for processing, as shown in Figure 6.12.[2].

When Message-parser identifies a received XML stream as AuthZ ticket, it parses and calls the SetEntryTVStable() or DeleteEntryTVStable() with parsed parameters to the TVS (Token Validation Service) plugin.

6.4.2 ForCEG – plugins

ForCEG is a software package written in Java that implements several functionalities such as WebService interface to outside world, ForCES interface down to hardware (network processors), and other specific functionalities (e.g., validation services for the XML stream received from outside world, connection to 3rd party system that is not ForCES compliant). These specific functionalities are designed and developed as ***plugins***. As illustrated in Figure 6.12, we have the following plugins:

- Message-parser (See Figure 6.12.[2]);
- TVS (Token Validation Service, as in Figure 6.12.[3]);
- Gmp (GMPLS gateway, as in Figure 6.12.[5]);
- APM (Advanced Power Management, as in Figure 6.12.[6]);

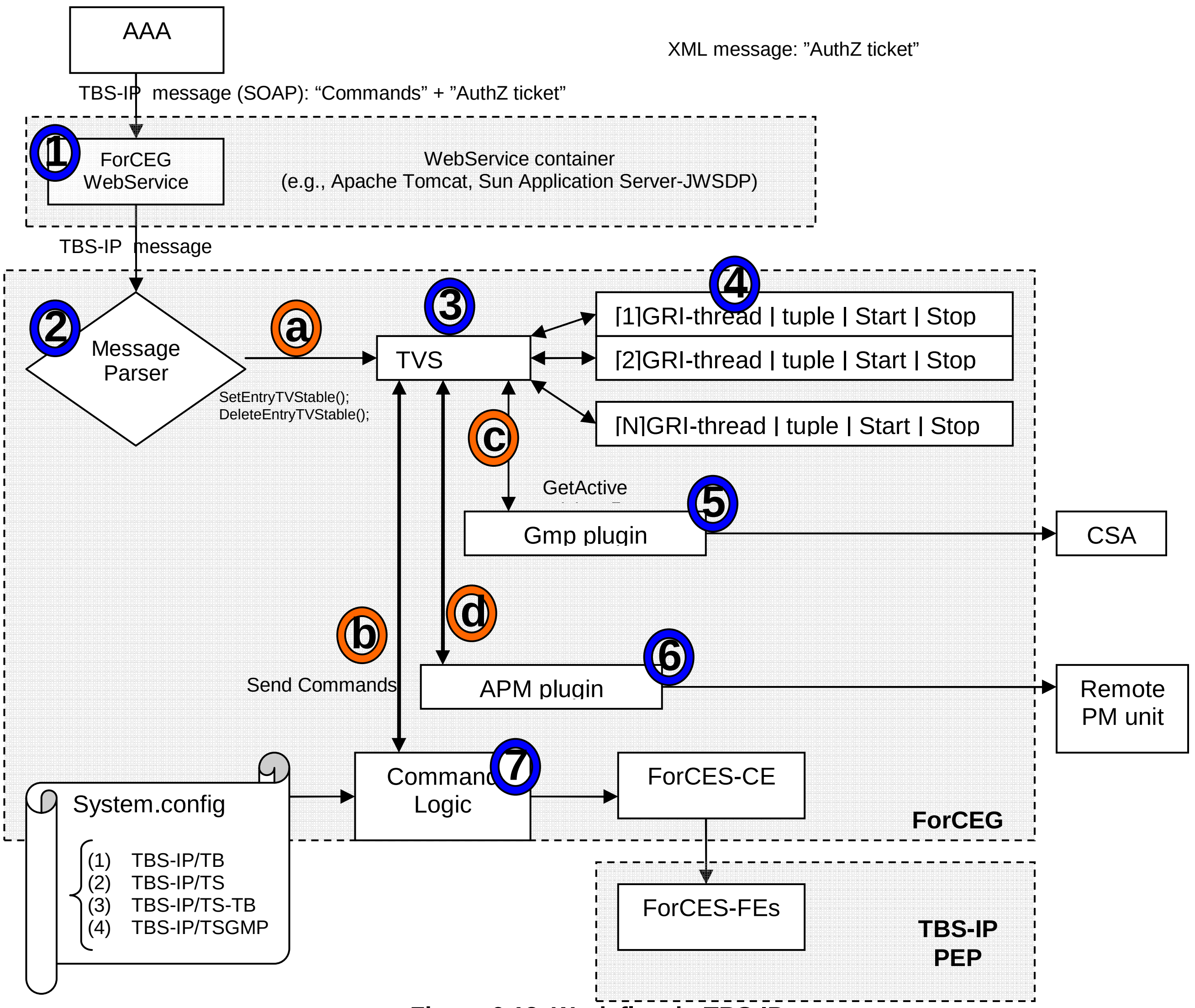
AAA
XML message: "AuthZ ticket"
TBS-IP message (SOAP): "Commands" + "AuthZ ticket"
1
ForCEG WebService
WebService container (e.g., Apache Tomcat, Sun Application Server-JWSDP)
TBS-IP message
2
Message Parser
a
SetEntryTVStable(); DeleteEntryTVStable();
3
TVS
4
[1]GRI-thread | tuple | Start | Stop
[2]GRI-thread | tuple | Start | Stop
[N]GRI-thread | tuple | Start | Stop
c
GetActive
5
Gmp plugin
CSA
b
d
Send Commands
6
APM plugin
Remote PM unit
System.config
(1) TBS-IP/TB
(2) TBS-IP/TS
(3) TBS-IP/TS-TB
(4) TBS-IP/TSGMP
Command Logic
7
ForCES-CE
ForCEG
ForCES-FEs
TBS-IP PEP

Figure 6.12. Work flow in TBS-IP router.

When the system starts up, the ***Command Logic*** (Figure 6.12.[7]) loads a system configuration file that specifies in which mode the TBS-IP router is going to run: (1) TB-IP/TB, (2) TBS-IP/TS, (3) TBS-IP/TS-TB, or (4) TBS-IP/TSGMP. By default, the TBS router requires the following plugins: ***Message Parser***, TVS, and ***APM***. Moreover, in case of the last option, the Command Logic also starts the ***gmp*** plugin such as to establish a connection between TBS-IP router and GMPLS systems (by connecting the ForCEG to CSA).

Each of plugins is part of the same Java package: ForCEG. The plugins are described in the following sections.

6.4.2.1 *Message-parser*

Message-parser (see Figure 6.12.[2]) is a gateway module that allows future extensions to the ForCEG with other plugins. Message-parser receives SOAP/XML messages; it parses the header of the message in order to classify it. For instance, if the message has a command "Add" and the body has an AuthZ ticket, then it extracts the useful fields from the ticket (e.g., GRI, StartTime, StopTime), and it calls "SetEntryTVSTable" to a TVS plugin, as shown in Figure 6.12.[a].

6.4.2.2 *Token Validation Service (TVS) plugin*

API:

Function	Called by module
Boolean SetEntryTVSTable(struct TUPLE, date StartTime, date StopTime);	Message-parser
Boolean SetEntryTVSTable(struct TUPLE);	Message-parser
Boolean RemoveEntryTVSTable(struct TUPLE);	Message-parser
Boolean AddPath(struct TUPLE);	GRI-thread
Boolean RemovePath(struct TUPLE);	GRI-thread
List(GRI) GetActiveGRI();	Gmp

The ***TVS*** plugin (the module shown Figure 6.12.[3]) is integrated within the ForCEG package and implements the following functionalities, as also illustrated in Figure 6.13:

- Wait for new commands (1):
  - o if a ticket came with a "*Remove*" command, then it searches through the GRI-thread list, issues the *RemovePath* command if it was an active GRI (such as the hardware also removes the active path), and then it removes itself (2).
  - o If a ticket came with an "*Add*" command, checks for NotValidBefore and NotValidAfter fields (3):
    - ▪ If it has no validation fields, then it is directly executes *AddPath* command without creating any thread (4);
    - ▪ When it has a validation field (specifies a future period of working time), TVS creates a new "GRI-thread" and it initialises the GRI-thread (5) with the following info: NotValidBefore, NotValidAfter, the TUPLE (LRI, Keys, AuthPort), and a reference to the parent module (TVS plugin). The reference is used for callbacks to parent;

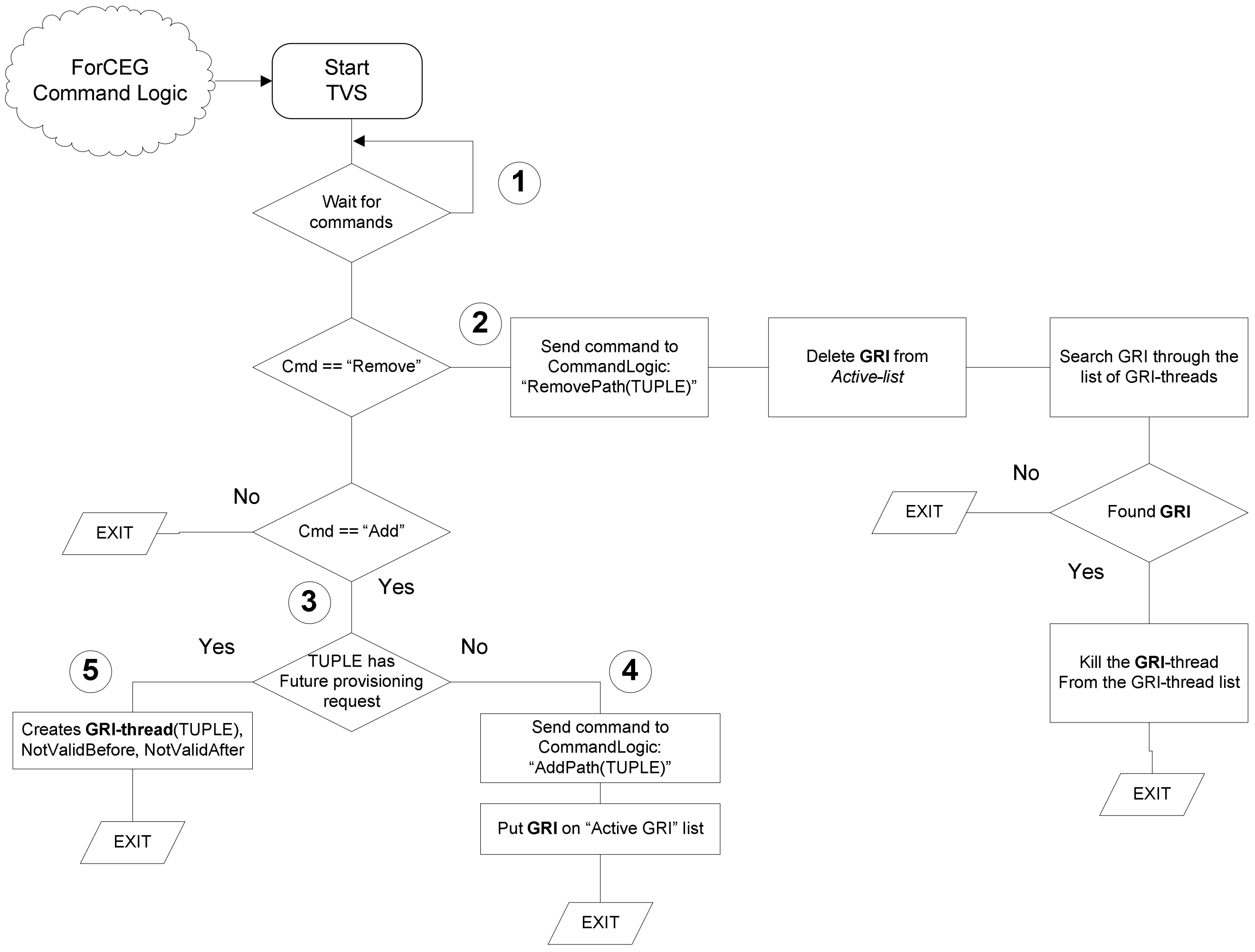
ForCEG Command Logic
Start TVS
1
Wait for commands
2
Cmd == "Remove"
Send command to CommandLogic: "RemovePath(TUPLE)"
Delete GRI from Active-list
Search GRI through the list of GRI-threads
No
EXIT
Found GRI
Yes
Kill the GRI-thread From the GRI-thread list
EXIT
No
EXIT
Cmd == "Add"
Yes
3
Yes
5
TUPLE has Future provisioning request
No
4
Creates GRI-thread(TUPLE), NotValidBefore, NotValidAfter
EXIT
Send command to CommandLogic: "AddPath(TUPLE)"
Put GRI on "Active GRI" list
EXIT

Figure 6.13. Workflow in TVS module.

- Sends commands to ForCEG on behalf of the GRI-thread (see Figure 6.12.[b]):
  - BOOL ReturnVal = AddPath(struct TUPLE);
  - This function also puts the GRI-thread on an "*Active GRI*" list.
  - BOOL ReturnVal = RemovePath(struct TUPLE);
  - This function removes the "GRI-thread" from the "*Active GRI*" list. It also executes the following checks for the power management module:
    - Computes the "*FirstWakeUp*" time (is the earliest time in future when a passive GRI-thread will send an AddPath command to hardware);
    - If (*FirstWakeUp > PM_Threshold)* then TVS sends "*STOP_HW*" command to the APM module, where *PM_Threshold* is the minimum time (e.g., 1 hour) while the hardware should be turned off for power saving reasons. As shown in Figure 6.12.[d], TVS sends commands to the APM module and receives the hardware status (ON/OFF) through the same interface (d).
  - Object AttributeVal = ReadLFBAttribute(string LFBname, string AttributeName);
  - The return object contains all values of the queried attribute. Used for checking of packet counters from Rx, Tx modules for statistic or debugging purposes.

- Process diagnostic messages received from lower levels (events from an LFB running on hardware):
  - SubscribeEvent(string LFBname, string AttributeName, int iCondition, int iThreshold);

- callback ProcessDiagnostic(string LFBname, string AttributeName, int iValue);

- Implements a power management function:
  - When TVS receives a notification signal from a passive GRI-thread, it checks if HW_STATUS==OFF then sends "*START_HW*" command to the APM module;

6.4.2.3 *Global Reservation Id threading (GRI-thread)*

A GRI-thread (Figure 6.12.[4]) is a class that runs in a separate thread container and it is instantiated by the TVS plugin. A GRI-thread has the following functionalities, as also illustrated in Figure 6.14:

1. Wait until (ValidNotBefore – *PM_StartUp)*, where *PM_StartUp* is a time needed by the power management unit to prepare the hardware (e.g., *PM_StartUp=5min)*;
2. Send a notification signal to the TVS module that this GRI needs the hardware ready (within *PM_StartUp* time),
3. Sleep for (*PM_StartUp)*, where PM_StartUp is the effective time needed to wake up the hardware;
4. Send AddPath(tuple) command to its parent (TVS plugin);
5. Wait until ValidNotAfter;
6. Send RemovePath(tuple) command to its parent;
7. Wait till proper acknowledge received when RemovePath() returns and then exit the process (kills itself);

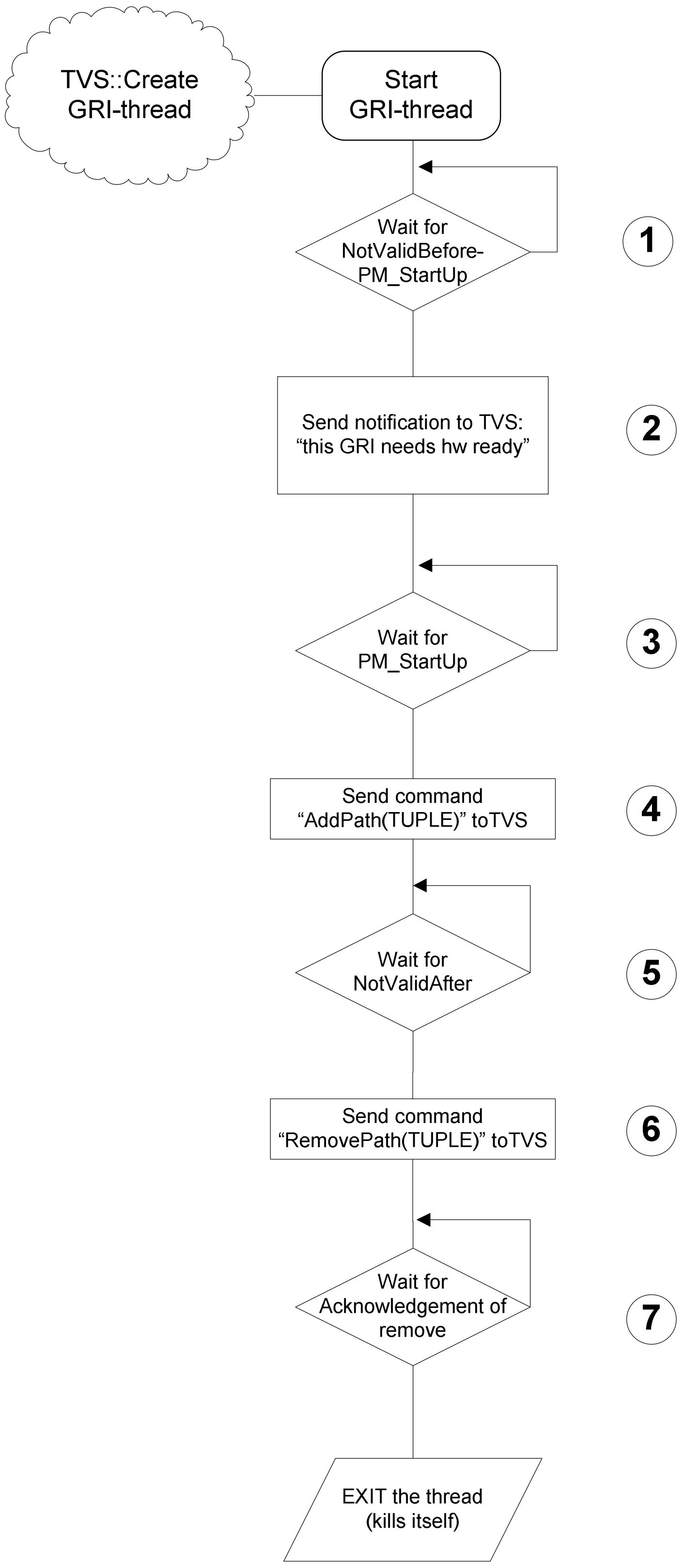
TVS::Create GRI-thread
Start GRI-thread
Wait for NotValidBefore-PM_StartUp
1
Send notification to TVS: "this GRI needs hw ready"
2
Wait for PM_StartUp
3
Send command "AddPath(TUPLE)" toTVS
4
Wait for NotValidAfter
5
Send command "RemovePath(TUPLE)" toTVS
6
Wait for Acknowledgement of remove
7
EXIT the thread (kills itself)

Figure 6.14. Workflow in GRI-thread module.

6.4.2.4 *GMPLS (Gmp) plugin*

API:

Function	Called by module
Boolean StartKeepAlive();	TVS
Boolean StopKeepAlive();	TVS

Gmp plugin (Figure 6.12.[5]) connects ForCEG to a 3rd party system: a GMPLS router. The GMPLS router runs a VLSR-client software called CSA. We design and implement an interface between ForCEG and CSA as a CE-CE interface (master-to-master).

The Gmp plugin has the following goals:

1. connects ForCEG to CSA at the start-up of ForCEG in case of loading of TBS-IP/TSGMP configuration;
2. disconnects ForCEG from CSA when ForCEG closes;
3. sends periodic messages to CSA: one message per path update (an entry in the AuthTable), or one message contains a batch of all active paths;
    a. When Timer ticks 10sec then search for active GRIs,
    b. take their iDs and build a message for CSA,
    c. and send the update message to CSA.

6.4.2.5 *Advance Power Management (APM) plugin*

API:

Function	Called by module
Boolean StartHW();	TVS
Boolean StopHW();	TVS
Boolean GetHWStatus():	TVS

APM plugin (Figure 6.12.[6]) is a software module that interconnects ForCEG to a hardware unit (e.g., APC MasterSwitch) that controls remotely the power socket relays in use by the network processors, routers, etc. The module interconnects to the hardware unit over snmp protocol.

6.4.3 **FFPF software on network processors: FIX2850**

FIX2850 [43] is a software project written in MicroC (ANSI-C language adapted specifically for parallel µEngine programming). It consists of multiple modules, each mapped on one µEngine. We need to build a flexible architecture that (re-)configures the installed LFBs onto the µEngines.

6.4.3.1 *FE-LFB in NPU: mapping of LFBs on hardware cores*

At loading time, the control code on XScale performs the LFB mapping onto available hardware cores (µEngines). Some LFBs are mapped, by design, on chosen µEngines, as follows:

- RX_Gig – running on µEngine$_0$ of the IngressNPU;
- Tx_CSIX – running on µEngine$_7$ of the IngressNPU;
- Rx_CSIX – running on µEngine$_0$ of the EgressNPU;
- Tx_Gig – running on µEngine$_7$ of the EgressNPU;

The custom application modules (TB, TS) can be mapped on any other available µEngine (e.g., µEngine$_1$, µEngine$_2$).
When FE is requested to map a certain application (e.g., TS, or TB), it locally instructs the NPU hardware how to map the required application. In other words, it takes the ForCES model of FEObject Location(LFB), as follows:

```
LFB("Rx")->Attribute("Location")->Value(0);
LFB("TS1")->Attribute("Location")->Value(1);
```

The LFB mapping onto µEngines is then described in the hardware by means of a description table (DT), as in the following example:

Table 1

Ingress NPU		**Egress NPU**	
LFB	**Location (µEngine)**	**LFB**	**Location (µEngine)**
RX	0	RX_csix	0
TX_csix	7	TX	7
TS_1	1	TS_2	1
TB_1	2	TB_2	2

6.4.3.2 *FE-LFB in NPU: mapping of processing hierarchy*

The LFB graph connection (processing hierarchy) is programmable by means of using specific "routing" code at the end of each packet processing dataflow within an LFB. This routing code decides where the current processed packet should go for further processing.

Figure 6.15 shows an example of graph interconnection when using one TB module per NPU. Therefore, the incoming traffic is first split in two (load balancing between those 2 NPUs): one half goes to the ingress TB_1 and the other half goes to the egress TB_2 via the local TX_csix module.

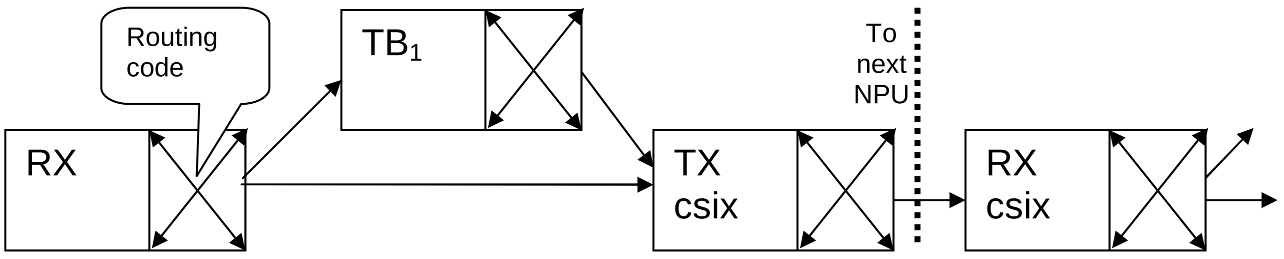

Figure 6.15. Workflow in fast data-path.

The routing code for each LFB in the given example in Figure 6.15 is illustrated in Figure 6.16 and then described as follows:

RX
```
if (pkt.index % 2)
    enqueue(TB1);
else
    enqueue(Tx);
```

TB_1
```
if (pkt has valid token)
    pkt.port =  authorized_port;
else
    pkt.port =  default_port;
enqueue(Tx);
```

TX_csix
```
send all pkts
to CSIX bus.
```

RX_csix
```
if (pkt has assigned port)
    enqueue(Tx);
else
   enqueue(TB2);
```

Figure 6.16. Code examples in decisional connectors.

The forwarding packet feature is expressed by the following line code: enqueue(LFB_{iD}). The flexibility of re-mapping of LFBs is given by the routing code itself. However, when using Intel IXP network processors,

changing the routing code requires re-compilation of the entire LFB code. In order to make a dynamic re-mapping of LFBs (assuming that all the required LFBs are available and loaded) we use the LFB_{iD} parameters. In other words, we model one LFBs connection in a dependency form:
“LFB_{iD}-src -> connector -> LFB_{iD}-dest”. Where the connector is one of the condition branch that when it is true activates the specific connector.

When FE is requested to map a certain application (e.g., TS, or TB), then it locally instructs the NPU hardware how to map the required processing hierarchy. In other words, it sets the ForCES attribute “Connector” on each LFB, as in the following example:

LFB(“Rx”)->Attribute(“Connector”)->[Value(0)], Value(1)];
LFB(“Rx”)->Attribute(“Connector”)->[Value(1), Value(7)];
LFB(“TB1”)->Attribute(“Connector”)->[Value(1), Value(7)];

A connection graph would be described by a table as follows:

Table 2

	LFB_src	connector	LFB_dest
Ingress_TB	0 (RX)	0	1 (TB_1)
	0	1	7 (Tx)
	1 (TB_1)	0	7
	1	1	7
Egress_TB	0	0	7
	0	1	1
	…		
	…		
	…		

This connection graph table will be loaded at start-up by a boot loader code from a shared Description Table (DT) into local registry on each µEngine. DT is filled in by the control code (FE-IXP) running on the XScale control processor.

6.4.3.3 *LFB attributes*

Each LFB has some attributes such as packet counters, diagnostic states, AuthTable. Each item uses certain data types such as DWORD for counters, BYTE for states, array of struct TUPLE for AuthTable. The LFB attributes should be addressable in a logic form such as:

FE(Ingress)->LFB(Rx)->Attribute(Pkts_count);
FE(Ingress)->LFB(TS1)->Attribute(TUPLE);

Example of attributes:

LFB	Attribute	Property	Type
Rx	Packet_Count Connector	Read-Write Read-Write	Integer Array (u8, u8)
Tx	Packet_Count Connector	Read-Write Read-Write	Integer Array (u8, u8)
Rx_CSIX	Packet_Count Connector	Read-Write Read-Write	Integer Array (u8, u8)
Tx_CSIX	Packet_Count Connector	Read-Write Read-Write	Integer Array (u8, u8)
TB	Proc_Packet_Count	Read-Write	Integer

	Connector Tuple	Read-Write Read-Write	Array (u8, u8) Array of TUPLE
TS	Proc_Packet_Count Connector Tuple Bad_Token_Count	Read-Write Read-Write Read-Write Read-Write	Integer Array (u8, u8) Array of TUPLE Integer

An example of complex LFB attributes is an AuthTable used by any TB or TS LFB. Each entry of the AuthTable is a TUPLE structure, defined as follows:

```
struct TUPLE
{
        char[20]        LRI;
        char[20]        LRI_NextDomain;
        char[20]        TokenKey;
        char[20]        TokenKey_NextDomain;
        u8    Port1;
        u8    Port2;
        char[20]        IPpacketMask;
        char[6]    IPsource;
        char[6]    IPdestination;
        char[4]    PortSource;
        char[4]    PortDestination;
        char    Status;
};
```

Summarising, an AuthTable looks as follows.

Table 3

Entry	0	1	2	3
LRI				
LRI_NextDomain				
TokenKey				
TokenKey_NextDomain				
Port1				
Port2				
IPpacketMask				
IPsource				
IPdestination				
PortSource				
PortDestination				
Status				

6.4.4 FE-IXP

FE-IXP is a cross-platform module that implements the first two layers of the ForCES (ForCES communication on top of TCP and ForCES modelling). FE-IXP runs as a deamon in userspace of the embedded linux OS.on an NPU hardware. It implements the ForCES standard in a FE that controls all LFBs within the current NPU and connects to the master CE located on a remote host. It mainly consists of the following layers:

- TCP server communication module with external host;
- ForCES model in FE<-> LFBs;
- Low-level connection (hardware abstraction layer – HAL) that is implemented in an external module: µeManager.

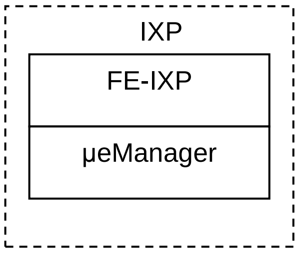

Figure 6.17

6.4.4.1 *FE*

FE is a module that implements ForCES protocol for FE (Forwarding Element), and is implemented in C++ and cross-compiled for ARM target (XScale control processor from IXP). Each IXP runs one FE that connects to its CE (Control Element) parent: ForCEG.

6.4.4.2 *µeManager*

µeManager governs the µEngines: it partitions the available DRAM memory, it assigns the memory to LFBs and it maps the LFBs onto µEngines by filling up the descriptor table (DT). µeManager also has to perform some hardware initialization such as MSF Unit. We choose for implementation as a userspace process running on the XScale processor of each IXP2850 NPU. This gives us more flexibility in what we can really do. We are not limited in memory allocation, execution of other process (like initialization scripts, compilers), network access etc. However, all control registers we need to interact with µEngines are available through /dev/mem device which gives direct access to physical memory space (with help of some external libraries for IXPs such as uengine and IXA_SDK). Since most of developers use Intel IXA_SDK for writing µEngine code, we can use this IXA library for uploading of the object code.

API:

Function	Called by module
Bool LFB_MapOnHw(string LFBname, int HW_CoreID);	FE
Bool LFB_MapHierarchy(int HW_CoreID_src, int HW_CoreID_dest,int condition);	FE
Bool LFB_AddPath(struct TUPLE);	FE
Bool LFB_RemovePath(struct TUPLE);	FE
Bool LFB_GetValue(string LFBName, string AttributeName);	FE

One important issue is the interaction with micro-code. Once the application is uploaded onto both NPUs, it needs to synchronize its start-up each other. This is accomplished by a “Start” signal generated by the host’s CE and received (almost) simultaneously on both NPUs from the unique host.

6.4.4.3 *Init_chip*

It configures the IXP2800 MSF's SPI-4.2, CSIX and flow control paths, and on the egress NPU, additionally configures the IXF1110 MAC.

ue_ixp_init.[c,h] – implements the HW initialization of each sub-component:
npu_reset(); npu_init_pre_clocks(); npu_init_clocks(); npu_enable_rx_tx(); npu_train_links();
ixf1110_init();

6.4.4.4 *Init_software*

It mainly performs the following operations:

- allocates the buffers (PBuf, IBufs for each µEngine, etc.);

- initialises the inter-communication paths (XScale <-> µEngines);
- loads the datapath code onto µEngines;
- starts the hardware units (µEngines);

6.4.4.5 *µEngines => XScale interface*

µEngine generates an interrupt for XScale.

Updating data from µEngine to XScale:

```
µEngines:
Packet comes in & token is valid:
        If (Status is 0) then {Status++; sends-to-XScale "Status";}
        Else {Status++; if (Status reaches MAX) then {Status=1; sends-to-XScale "Status";} }
If update signal received from XScale: Perform the requested operation such as update a
certain status.
OBS. MAX gives the dead-time between µEngines –to- XScale messages.
```

```
XScale:
When Timer1=10secs:
        Check every Status[] from the list and for each Status not 0 => sends up the GMPLS
update message;
When Timer2=3secs:
        Check every Status ? Status_old:
  If Status = = Status_old: sends a time-out message to µEngines to clear the Status;
  If Status <> Status_old: Status_old=Status.
```

6.4.4.6 *XScale => µEngines interface*

The interface uses shared memory to send messages on demand with the help of signals. It uses Interthread signal: XScale sends the Interthread_ME signal to a certain thread in a certain µEngine.
For instance, when XScale receives a request to add/remove a path, it first updates the shared memory (SDRAM, or SRAM) and then sends a signal to the µEngine that runs the specific LFB (e.g., TS, or TB).

6.4.4.7 *Mapping of high-level identifiers into low-level hardware*

As described before, the authorisation table (AuthTable) contains some large fields such as LRI, IPpacketMask, etc. However, given the hardware constrains in memory, we need to provide to the µEngines only the strict fields needed in runtime. Therefore, the TUPLE is going to be mapped into MINI_TUPLE, as follows:

```
typedef struct _TUPLE
{
  u_int8_t   LRI[20];
  u_int8_t   LRI_NextDomain[20];
  u_int8_t   TokenKey[20];
  u_int8_t   TokenKey_NextDomain[20];
  u_int8_t   Port1;
  u_int8_t   Port2;
  u_int8_t   IPpacketMask[20];
  u_int8_t   IPsource[4];
  u_int8_t   IPdestination[4];
  u_int8_t   PortSource[2];
  u_int8_t   PortDestination[2];
  u_int8_t   Status;
}TUPLE;
```

```
typedef struct _MINI_TUPLE
{
  u_int32_t   iD;
  u_int32_t   TokenKey[5];
  u_int32_t   TokenKeyNextDomain[5];
  u_int32_t   Port1;
  u_int32_t   Port2;
  u_int32_t   Status;
}MINI_TUPLE;
```

The Authorization table is composed of MINI_TUPLE entries and it resides in the slowest memory (and the smallest): LocalMem, or SRAM.

Table 4

Entry	iD [4Bytes]	TokenKey [20Bytes]	TokenKey_NextDomain [20Bytes]	Port1 [4Bytes]	Port2 [4Bytes]	Status [4Bytes]
0	…					
1	…					
2	…					

The mapping of one TUPLE into MINI_TUPLE is done at each AddPath() call. It consists of computing a small identifier iD as follows:

iD = Hash64(IPpacketMask over the IPsource, IPdestination, PortSource, PortDestination);

The other fields remain same: TokenKey, TokenKey_NextDomain, Port1, Port2.
We have chosen to use Hash64 because it is hardware supported and a fast function to provide (cvasi)unique identifier over a large set of data. Note that the same hash function will be performed in datapath for each incoming packet in order to find the unique iD for a searching in the AuthTable in order to retrieve the proper entry.

7 Conclusion

This deliverable describes the result of the development of the AAA Authorisation infrastructure for multidomain Optical Network Resources Provisioning (ONRP). The proposed architecture attempts to address key access control problems when integrating heterogeneous Network Resource Provisioning Systems (NRPS) being deployed in the different Phosphorus testbeds. The proposed architecture also targets to ensure future compatibility with the Grid and NREN access control solutions and infrastructures.

The report summarises recent research and developments of the Generic AAA Authorisation framework (GAAA-AuthZ) to support Complex Resource Provisioning (CRP) and provides general implementation suggestions and design recommendations to the GAAA-AuthZ for ONRP which were used to compile extensive list of detailed requirements to different components of the developed GAAA-AuthZ infrastructure

The document provides a general overview of the WP1 Network Service Plane (NSP) AAA/AuthZ issues, discusses how GAAA-AuthZ can be implemented in the G.OUNI/G2MPLS, and provides an overview of the eduGAIN framework that is considered as a recommended common Authentication and Authorisation Infrastructure (AAI) for interdomain network resource provisioning between European NREN's.

The extensive list of detailed technical requirements to the GAAA-AuthZ infrastructure for ONRP presented in section 3 summarises available experience among WP4 partners and provides a solid basis for GAAA-AuthZ development roadmap.

The proposed GAAA-AuthZ architecture introduces a number of mechanisms and solutions to support multidomain ONRP such as: AuthZ ticket format for extended AuthZ session management, Token Validation Service (TVS) to enable token based policy enforcement, policy Obligation Handling Reference Model (OHRM), and XACML policy profile for OLPP. The proposed architecture will allow smooth integration with other AuthZ frameworks as currently used and developed by NREN and Grid community.

The described implementation of the GAAA-AuthZ components in the framework of the GAAA Toolkit supports major AAA/AuthZ functions. It allows easy integration of the AAA/AuthZ service into existing NRPS frameworks and provides a basis for further development and extensions. In particular, TVS and GAAAPI implementation allow their integration into the Token Based ForCES Switch and DRAGON/OSCARS NRPS.

The report provides detailed technical information about the current development and implementation of the Token Based Networking (TBN) and ForCES architecture which is being developed in cooperation between the University of Amsterdam and the University of Patras as a part of the WP4 activity.

It is intended that this deliverable will provide a technological and technical basis for adding AAA/AuthZ services to ONRP technologies and components being developed by other Phosphorus packages. Proposed GAAA-AuthZ solutions and current implementation in GAAA Toolkit can be used in basic network provisioning scenarios and frameworks such as NRPS and Grid GMPLS.

Based on current GAAA-AuthZ architecture development and GAAA Toolkit implementation, WP4 will focus on extending GAAA Toolkit functionality and assisting other WP's in integrating AAA/AuthZ services into target NRPS's. This should also provide feedback for further architecture update and GAAA Toolkit development.

Particular technical development will include implementation of the proposed XACML policy profile for OLPP, Obligation Handling Reference Model (OHRM). This work will also require wider cooperation with the Grid and application security community that have similar tasks in developing dynamic AuthZ services for on-demand Complex Resource Provisioning to propose common standardisation framework.

8 References

[1] RFC2903 Laat de, C., G. Gross, L. Gommans, J. Vollbrecht, D. Spence, "Generic AAA Architecture," Experimental RFC 2903, Internet Engineering Task Force, August 2000. - ftp://ftp.isi.edu/in-notes/rfc2903.txt

[2] RFC 2904 - "AAA Authorization Framework" J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, August 2000. - ftp://ftp.isi.edu/in-notes/rfc2904.txt

[3] ITU-T Rec. X.811 (1995) | ISO/IEC 10181-3:1996, Information technology - Open systems interconnection - Security frameworks in open systems: Access control framework. - http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.812-199511-I!!PDF-E&type=items

[4] ITU-T Rec. X.812 (1995) | ISO/IEC 10181-2:1996, Information technology - Open systems interconnection - Security frameworks in open systems: Authentication framework. - http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.811-199504-I!!PDF-E&type=items

[5] GFD.38 Conceptual Grid Authorization Framework and Classification. M. Lorch, B. Cowles, R. Baker, L. Gommans, P. Madsen, A. McNab, L. Ramakrishnan, K. Sankar, D. Skow, M. Thompson. - http://www.ggf.org/documents/GWD-I-E/GFD-I.038.pdf

[6] Gommans, L. et al, "Applications Drive Secure Lightpath Creation across Heterogeneous Domains", Special Issue "*IEEE Communications Magazine, Feature topic Optical Control Planes for Grid Networks: Opportunities, Challenges and the Vision*", March 2006.

[7] Demchenko Y., L. Gommans, C. de Laat, "Using SAML and XACML for Complex Authorisation Scenarios in Dynamic Resource Provisioning", in Proc. *The Second International Conference on Availability, Reliability and Security (ARES 2007)*, Vienna, Austria, April 10-13, 2007. IEEE Computer Society, ISBN: 0-7695-2775-2, pp. 254-262.

[8] Demchenko, Y., Leon Gommans, Cees de Laat, Rene van Buuren, "Domain Based Access Control Model for Distributed Collaborative Applications", Proceedings of The 2nd IEEE International Conference on e-Science and Grid Computing, December 4-6, 2006, Amsterdam.

[9] Y. Demchenko, et al., Dynamic security context management in Grid-based applications, Future Generation Computer Systems (2007), doi:10.1016/j.future.2007.07.015

[10] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker, „Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)“, OASIS Standard Specification, 1 February 2006

[11] OIF-UNI-01.0. User Network Interface (UNI) 1.0 Signalling Specification, October 1, 2001.- http://www.oiforum.com/public/documents/OIF-UNI-01.0.pdf

[12] Phosphorus Deliverable D.2.7: Grid-GMPLS network interfaces specification. -

[13] RFC2748: The COPS (Common Open Policy Service) Protocol, Edited Durham, D., January 2000. - http://www.ietf.org/rfc/rfc2748.txt

[14] RFC2753: A Framework for Policy-based Admission Control, January 2000. - http://www.ietf.org/rfc/rfc2753.txt

[15] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu, A Usage-based Authorization Framework for Collaborative Computing Systems, in the proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT), 2006.

[16] "Token-based authorization of connection oriented network resources", by Leon Gommans, Franco Travostino, John Vollbrecht, Cees de Laat, and Robert Meijer, in Proceedings of GRIDNETS, San Jose, CA, USA, Oct 2004.

[17] Deliverable DJ5.2.2: GÉANT2 Authorisation and Authentication Infrastructure (AAI) Architecture. - http://www.geant2.net/upload/pdf/GN2-05-192v6.pdf

[18] Deliverable DJ5.2.3,2: Best Practice Guide - AAI Cookbook - Second Edition Guidelines for Connecting to the eduGAIN AA Infrastructure. - http://www.geant2.net/upload/pdf/GN2-07-023v4-DJ5-2-3_2_Best_Practice_Guide-AAI_Cookbook-Second_Edition.pdf

[19] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell’Agnello, A. Frohner, A. Gianoli, K. L˝orentey, and F. Spataro, “VOMS, an Authorization System for Virtual Organizations”. - http://grid-auth.infn.it/docs/VOMS-Santiago.pdf

[20] Shibboleth Project. - http://shibboleth.internet2.edu/

[21] InCommon Federation - http://www.incommonfederation.org/

[22] TCG Infrastructure Working Group Reference Architecture for Interoperability (Part I). Specification Version 1.0, Revision 1. 16 June 2005. [Online]. Available: I - https://www.trustedcomputinggroup.org/specs/IWG/IWG_Architecture_v1_0_r1.pdf

[23] Chadwick, D., "Use of WS-TRUST and SAML to access a CVS". OGSA-AUTHZ WG Draft. [Online]. Available: https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-authz/docman.root.authz_service/doc9011/1

[24] Web Services Trust Language (WS-Trust) - ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf

[25] Viola Meta Scheduling Service Project. - http://packcs-e0.scai.fhg.de/viola-project/

[26] GFD.107, WS-Agreement Specification V1.0. - http://www.ggf.org/documents/GWD-I-E/GFD-I-E.038.pdf

[27] "XACML 3.0 administrative policy," OASIS Draft, 10 December 2005. [Online]. Available from http://docs.oasis-open.org/access_control

[28] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS Standard, 15 March 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[29] SAML 2.0 Profile of XACML 2.0, Version 2. Working Draft 2, 26 June 2006. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/xacml-2.0-profile-saml2.0-v2.zip

[30] *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS Standard, 1 February 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[31] Information Technology - Role Based Access Control, Document Number: ANSI/INCITS 359-2004, InterNational Committee for Information Technology Standards, 3 February 2004, 56 p

[32] "Multiple resource profile of XACML 2.0", OASIS Standard, 1 February 2005, available from http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-mult-profile-spec-os.pdf

[33] "Hierarchical resource profile of XACML 2.0", OASIS Standard, 1 February 2005, available from http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-hier-profile-spec-os.pdf

[34] *Core and hierarchical role based access control (RBAC) profile of XACML v2.0*, OASIS Standard, 1 February 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf

[35] ESnet On-Demand Secure Circuits and Advance Reservation
System (OSCARS). - http://www.es.net/oscars/

[36] The DRAGON Project. - https://dragon.maxgigapop.net/twiki/bin/view/DRAGON/WebHome

[37] Menezes A., P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography". - ISBN: 0-8493-8523-7, October 1996, 816 pages

[38] A. Shamir. Identity-based cryptosystems and signature schemes. In G.R. Blakley and D. Chaum, editors, Advances in Cryptology - Proceedings of CRYPTO'84, pages 47{53. Springer-Verlag LNCS 196, 1985.

[39] H. Tanaka. A realization scheme for the identity-based cryptosystem. In C. Pomerance, editor, Advances in Cryptology - Proceedings of CRYPTO'87, pages 340{349. Springer-Verlag LNCS 293, 1988.

[40] Generic Authorization Authentication and Accounting. [Online]. Available: http://www.science.uva.nl/research/ air/projects/aaa/

[41] Aaauthreach Project: GAAAPI implementation. - http://staff.science.uva.nl/~demch/projects/aaauthreach/index.html

[42] Evangelos Haleplidis, Robert Haas, Spyros Denazis, Odysseas Koufopavlou, "A Web Service- and ForCES-based Programmable Router Architecture", IWAN2005, France.

[43] Joel Halpern, Elen Deleganes, "ForCES Forwarding Element Model", IETF draft, work in progress, October 2007. - http://ietfreport.isoc.org/all-ids/draft-ietf-forces-model-08.txt

[44] "The Token Based Switch: Per-Packet Access Authorisation to Optical Shortcuts", by Mihai-Lucian Cristea, Leon Gommans, Li Xu, and Herbert Bos, in Proceedings of IFIP Networking, Atlanta, GA, USA, May 2007.

[45] "Applications drive secure lightpath creation across heterogeneous domains", by Leon Gommans, Freek Dijkstra, Cees de Laat, Arie Taal, Alfred Wan, Inder Monga, Franco Travostino, in IEEE Communications Magazine 44(3), March 2006.

Appendix A Acronyms

AAA	Authentication, Authorisation, Accounting
AAI	Authentication, Authorization Infrastructure
ACL	Access Control List
ASM	Application Specific Module (as part of the GAAA-AuthZ architecture)
AuthZ	Authorization
AuthN	Authentication
BoD	Bandwidth on-Demand
COPS	Common Open Policy Service Protocol Overview
CRP	Complex Resource Provisioning
CVS	Credential Validation Services
DAA	Direct Anonymous Attestation
DAC	Discretionally Access Control
DCAS	Domain Site Central Authorisation Service
DDSS	Distributed Data Storage Systems
DRAGON	Dynamic Resource Allocation over GMPLS Optical Networks
e2e	end to end
EGEE	Enabling Grids for E-sciencE (European Grid Project)
FC	Fibre Channel
ForCES	Forwarding and Control Element Separation
FPP	
GAAA-AuthZ	Generic AAA Authorisation Framework
GAAAPI	Generic Authentication/Authorisation Application Programming Interface
GEANT2	Pan-European Gigabit Research Network
gJAF	gLite Java Authorisation Framework
gLite	EGEE Grid middleware
GMPLS	Generalized MPLS (MultiProtocol Label Switching)
GSI	Grid Security Infrastructure
GT4	Globus Toolkit Version 4 (Web-Service based)
GT4-AuthZ	Globus Toolkit Authorisation Framework
ICC	
IdM	Identity Manager
IdP	Identity Provider
IKE	Internet Key Exchange

LFB	**Logical Function Block**
MAC	**Mandatory Access Control**
NREN	**National Research and Education Network**
NRP	**Network Resource Porvisioning**
OLPP – Optical LightPath Provisioning	
NRPS	**Network Resource Provisioning System**
OHRM	**Obligation Handling Reference Model**
OSCARS	**On-demand Secure Circuits and Advance Reservation System**
PAP	**Policy Authority Point**
PDP	**Policy Decision Point**
PEP	**Policy Enforcement Point**
PIP	**Policy Information Point**
PKC	**X.509 Public Key Certificate**
PKI	**Public Key Infrastructure**
PoP	**Point of Presence**
RBE	**Rule Based Engine**
QoS	**Quality of Service**
SAAS	**Shibboleth Attribute Authority Service**
SAML	**Security Assertion Markup Language**
SASL	**Simple Authentication and Security Layer**
SCAS	**Site Central Authorisation Service**
SNMP	**Simple Network Management Protocol**
SPKI	**Simple Public Key Infrastructure ((RFC 2692 and RFC 2693))**
S-R-A (-E)	**Subject – Resource – Action (- Environment) in relation to the XACML policy and context definition**
SSO	**Single Sign-On**
SSL	**Secure Socket Layer**
STS	**WS-Trust Secure Token Service**
TBN	**Token Based Networking**
TBS	**Token Based Switch**
TBS-IP	**TBS over IP traffic**
TB	**Token Builder**
TCG	**Trusted Computing Group**
TS	**Token Switch**
TLS	**Transport Layer Security**
TVS	**Token Validation Service**
VLSR	**Virtual Label Switching Routing**
VO	**Virtual Organisation**
VOMS	**Virtual Organization Membership Service**
UNI	**User to Network Interface**
UNICORE	**European Grid Middleware (UNIiform Access to COmpute REsources)**
VLAN	**Virtual LAN (as specified in IEEE 802.1p)**
VIOLA	**A German project funded by the German Federal Minitry of Education and Research (Vertically Integrated Optical Testbed for Large Applications in DFN)**
VPN	**Virtual Private Network**
WSAG	**Web Services Agreement**

XACML	**eXtensible Access Control Markup Language**
XKMS	**XML Key Management Specification**
XML	**eXtensible Markup Language**

Appendix B XML Ticket and Token Examples

B.1 AuthzTicket example

The listing below provide an example with the Resource element as required for TBS programming generated with the GAAAPI package. The listing also contains comments that explain a suggested mapping to SAML2.0 Authorisation assertion elements, which demonstrates that even for basic AuthZ session data, few extension elements are required for extended security context expression.

```
<AAA:AuthzTicket xmlns:AAA="http://www.aaauthreach.org/ns/#AAA" Issuer="urn:cnl:trust:tickauth:pep"
TicketID="cba06d1a9df148cf4200ef8f3e4fd2b3" SessionID=" f8f3e4fd2b3 df148cf4200">
  <AAA:Decisions><AAA:Decision ResourceID="http://resources.collaboratory.nl/Philips_XPS1"
result="Permit"/>
  </AAA:Decisions>
  <!-- SAML mapping: <AuthorizationDecisionStatement Decision="*" PolicyRef="*" Resource="*"> -->
<AAA:Resources><AAA:Resource ResourceID="http://www.aaaarch.org/TBS/ForceG">
  <AAA:LRI purpose="String">text</AAA:LRI>
  <AAA:TokenKey>String</AAA:TokenKey>
  <AAA:Ports>
    <AAA:port1>String</AAA:port1>
    <AAA:port2>String</AAA:port2>
  </AAA:Ports>
  <AAA:ApplicationFlow>
    <AAA:IPpacketMask>String</AAA:IPpacketMask>
    <AAA:IPsource>String</AAA:IPsource>
    <AAA:IPdestination>String</AAA:IPdestination>
    <AAA:PortSource>String</AAA:PortSource>
    <AAA:PortDestination>String</AAA:PortDestination>
  </AAA:ApplicationFlow>
</AAA:Resource></AAA:Resources>
  <AAA:Actions>
    <AAA:Action>cnl:actions:CtrlInstr</AAA:Action>     <!-- SAML mapping: <Action> -->
    <AAA:Action>cnl:actions:CtrlExper</AAA:Action>
  </AAA:Actions>
  <AAA:Subject Id="subject">
    <AAA:SubjectID>WHO740@users.collaboratory.nl</AAA:SubjectID>
    <!-- SAML mapping: <Subject>/<NameIdentifier> -->
    <AAA:SubjectConfirmationData>
         IGhA11vwa8YQomTgB9Ege9JRNnld84AggaDkOb5WW4U=</AAA:SubjectConfirmationData>
    <!-- SAML mapping:  EXTENDED <SubjectConfirmationData/> -->
    <AAA:Role>analyst</AAA:Role>
    <!-- SAML mapping:
             <Evidence>/<Assertion>/<AttributeStatement>/<Assertion>/<Attribute>/<AttributeValue> -->
    <AAA:SubjectContext>CNL2-XPS1-2005-02-02</AAA:SubjectContext>
    <!-- SAML mapping:
         <Evidence>/<Assertion>/<AttributeStatement>/<Assertion>/<Attribute>/<AttributeValue> -->
  </AAA:Subject>
  <AAA:Delegation MaxDelegationDepth="3" restriction="subjects">
  <!-- SAML mapping:  LIMITED <AudienceRestrictionCondition> (SAML1.1),
                                 or <ProxyRestriction>/<Audience> (SAML2.0)  -->
    <AAA:DelegationSubjects>
      <AAA:SubjectID>team-member-2</AAA:SubjectID>
      <AAA:SubjectID>team-member-1</AAA:SubjectID>
    </AAA:DelegationSubjects>
  </AAA:Delegation>
  <AAA:Conditions NotBefore="2006-06-08T12:59:29.912Z"
                       NotOnOrAfter="2006-06-09T12:59:29.912Z" renewal="no">
  <!-- SAML mapping: <Conditions NotBefore="*" NotOnOrAfter="*"> -->
    <AAA:ConditionAuthzSession PolicyRef="PolicyRef-GAAA-RBAC-test001" SessionID="JobXPS1-2006-001">
    <!-- SAML mapping: EXTENDED <SAMLConditionAuthzSession PolicyRef="*" SessionID="*"> -->
      <AAA:SessionData>put-session-data-Ctx-here</AAA:SessionData>
      <!-- SAML mapping:  EXTENDED <SessionData/> -->
    </AAA:ConditionAuthzSession>
  </AAA:Conditions>
  <AAA:Obligations>
    <AAA:Obligation>put-policy-obligation(2)-here</AAA:Obligation>
    <!-- SAML mapping:  EXTENDED <Advice>/<PolicyObligation> -->
    <AAA:Obligation>put-policy-obligation(1)-here</AAA:Obligation>
  </AAA:Obligations>
</AAA:AuthzTicket>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo> ... </ds:SignedInfo>
  <ds:SignatureValue>e4E27kNwEXoVdnXIBpGVjpaBGVY71Nypos...</ds:SignatureValue>
</ds:Signature>
```

Figure B.1. Example of XML based AuthZ ticket format used for TVS programming. (Note. Comments refer to the suggested SAML2.0 mapping)

B.2 XML Token example

XML token is used to provide a reference to already reserved/authorised resource at the access/usage stage of the reserved service or resource. Typically token use is supported by the detailed reservation information stored/cached at the resource.

XML token format uses a special “TVS/TBN” profile of the more general AuthzToken format. Example of the full TVS XML token is shown below:

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
        Issuer="urn:aaa:gaaapi:token:TVS"
        SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
        TokenId="d1384ab54bd464d95549ee65cb172eb7">
    <AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
    <AAA:Conditions NotBefore="2007-08-12T16:00:29.593Z" NotOnOrAfter="2007-08-13T16:00:29.593Z"/>
</AAA:AuthzToken>
```

where the element <TokenValue> and attributes SessionId and TokenId are mandatory, and the element <Conditions> and attributes Issuer, NotBefore, NotOnOrAfter are optional.

Minimum token format is illustrated in the following example:

```
<AAA:AuthzToken xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
        SessionId="a9bcf23e70dc0a0cd992bd24e37404c9e1709afb"
        TokenId="d1384ab54bd464d95549ee65cb172eb7">
     <AAA:TokenValue>ebd93120d4337bc3b959b2053e25ca5271a1c17e</AAA:TokenValue>
```

```
</AAA:AuthzToken>
```

Attributes “Issuer” allow for distinguishing different AuthzToken profiles. The TVS profile is identified by the URN "urn:aaa:gaaapi:token:TVS".

Appendix C Obligations definition in XACML policy

C.1 Obligations definition and schema

From XACML 2.0 specification section “2.12. Actions performed in conjunction with enforcement” (lines 557-566):

In many applications, policies specify actions that MUST be performed, either instead of, or in addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94]. XACML provides facilities to specify actions that MUST be performed in conjunction with policy evaluation in the `<Obligations>` element. This idea was described as a provisional action by Kudo [Kudo00]. There are no standard definitions for these actions in version 2.0 of XACML. Therefore, bilateral agreement between a *PAP* and the *PEP* that will enforce its *policies* is required for correct interpretation. *PEPs* that conform with v2.0 of XACML are required to deny *access* unless they understand and can discharge all of the `<Obligations>` elements associated with the *applicable policy*. `<Obligations>` elements are returned to the *PEP* for enforcement.

[XACML 5.45.] Element <Obligation>

The `<Obligation>` element SHALL contain an identifier for the *obligation* and a set of *attributes* that form arguments of the action defined by the *obligation*. The `FulfillOn` attribute SHALL indicate the *effect* for which this *obligation* must be fulfilled by the *PEP*.

```
<xs:element name="Obligation" type="xacml:ObligationType"/>
<xs:complexType name="ObligationType">
<xs:sequence>
<xs:element ref="xacml:AttributeAssignment" minOccurs=”0” maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
```

```
<xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
</xs:complexType>
```

The `<Obligation>` element is of **ObligationType** complexType. See Section 7.14 for a description of how the set of ***obligations*** to be returned by the ***PDP*** is determined. The `<Obligation>` element contains the following elements and attributes:

ObligationId [Required]

Obligation identifier. The value of the ***obligation*** identifier SHALL be interpreted by the ***PEP***.

FulfillOn [Required]

The ***effect*** for which this ***obligation*** must be fulfilled by the ***PEP***.

<AttributeAssignment> [Optional]

Obligation arguments assignment. The values of the ***obligation*** arguments SHALL be interpreted by the ***PEP***.

[XACML-5.46.] Element <AttributeAssignment>

The `<AttributeAssignment>` element is used for including arguments in *obligations*. It SHALL contain an `AttributeId` and the corresponding *attribute* value, by extending the AttributeValueType type definition. The `<AttributeAssignment>` element MAY be used in any way that is consistent with the schema syntax, which is a sequence of `<xs:any>` elements. The value specified SHALL be understood by the *PEP*, but it is not further specified by XACML. See Section 7.14. Section 4.2.4.3 provides a number of examples of arguments included in *obligations*.

```
<xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
<xs:complexType name="AttributeAssignmentType" mixed="true">
<xs:complexContent>
<xs:extension base="xacml:AttributeValueType">
<xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

The `<AttributeAssignment>` element is of **AttributeAssignmentType** complex type. The `<AttributeAssignment>` element contains the following attributes:
`AttributeId` [Required]
The ***attribute*** Identifier.

[XACML-7.14. Obligations]

A policy or policy set may contain one or more obligations. When such a policy or policy set is evaluated, an obligation SHALL be passed up to the next level of evaluation (the enclosing or referencing policy, policy set or authorization decision) only if the effect of the policy or policy set being evaluated matches the value of the FulfillOn attribute of the obligation.

As a consequence of this procedure, no obligations SHALL be returned to the PEP if the policies or policy sets from which they are drawn are not evaluated, or if their evaluated result is "Indeterminate" or "NotApplicable", or if the decision resulting from evaluating the policy or policy set does not match the decision resulting from evaluating an enclosing policy set.

If the PDP's evaluation is viewed as a tree of policy sets and policies, each of which returns "Permit" or "Deny", then the set of obligations returned by the PDP to the PEP will include only the obligations associated with those paths where the effect at each level of evaluation is the same as the effect being returned by the PDP. In situations where any lack of determinism is unacceptable, a deterministic combining algorithm, such as ordered-deny-overrides, should be used.

C.2 Obligations expression examples

a) Example <Obligations> from XACML2.0 specification:

```
<Obligations>
<Obligation ObligationId="urn:oasis:names:tc:xacml:example:obligation:email" FulfillOn="Permit">
<AttributeAssignment
   AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
   DataType="http://www.w3.org/2001/XMLSchema#string">
     &lt;AttributeSelector
     RequestContextPath="//md:/record/md:patient/md:patientContact/md:email"
     DataType="http://www.w3.org/2001/XMLSchema#string"/&gt ;
</AttributeAssignment>
<AttributeAssignment
   AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
   DataType="http://www.w3.org/2001/XMLSchema#string">
     Your medical record has been accessed by:
</AttributeAssignment>
<AttributeAssignment
   AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
   DataType="http://www.w3.org/2001/XMLSchema#string">
     &lt;SubjectAttributeDesignator
     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
     DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
</AttributeAssignment>
</Obligation>
</Obligations>
</Policy>
```

b) Example with pool accounts mapping in Grid

Note. Below options are provide only illustration how Obligations can be expressed in the XACML2.0 compliant format. This is not a goal of this document and section to make suggestions about preferred format, however it can be suggested that although Option 3 can bring possible flexibility it is not recommended way of expressing and communicating Obligations because of security concerns.

```
<!-- Obligations format option 1 (UID, GID explicitly mentioned as separate XML elements inside
AttributeAssignment element) -->
<Obligations>
<Obligation
   ObligationId="http://glite.egee.org/JRA1/Authz/XACML/obligation/map.poolaccount"
   FulfillOn="Permit">

<!-- This is a common part that specify to what kind of attribute the next ‘map.to’ action is applied
to -->
```

```
<AttributeAssignment
   AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute: requesting-subject"
   DataType="http://www.w3.org/2001/XMLSchema#string">
     &lt;SubjectAttributeDesignator
     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
     DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
</AttributeAssignment>

<!-- This is actual account attribute/name to which it should be mapped -->
<AttributeAssignment
   AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mapto"
   DataType="http://www.w3.org/2001/XMLSchema#string">
     &lt;UnixId
     DataType="http://www.w3.org/2001/XMLSchema#string"&gt;
          okoeroo&gt;UnixId&gt;
     &lt; GroupPrimary
     DataType="http://www.w3.org/2001/XMLSchema#string"&gt;
         computergroup&gt;GroupPrimary&gt;
     &lt;GroupSecondary
     DataType="http://www.w3.org/2001/XMLSchema#string"&gt;
         datagroup&gt;GroupSecondary&gt;
</AttributeAssignment>
</Obligation>
</Obligations>
```

Appendix D XACML Policy examples

D.1 XACML policy example with obligations and corresponding request/response messages

a) Policy example using simple conventions that require minimum PEP functionality.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
      xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
      xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
      xmlns:md="http://www.medico.com/schemas/record"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
        access_control-xacml-2.0-policy-schema-os.xsd"
      PolicyId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>
  Example007 - Test for policy selection by PEP type.
  </Description>
  <PolicyDefaults>
    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
  </PolicyDefaults>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch
           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
        VO-EGEE</AttributeValue>
        <SubjectAttributeDesignator
          SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject:-category:access-subject"
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-vo"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Actions>
      <AnyAction/>
    </Actions>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">http://nikhef.nl/VO-EGEE/CE01
        </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            DataType="http://www.w3.org/2001/XMLSchema#URI"/>
       </ResourceMatch>
    ResourceMatch
          MatchId=”urn:oasis:names:tc:xacml:1.0:function:string-equal”>
          <AttributeValue
    DataType= “http://www.w3.org/2001/XMLSchema#string”>GT4-CE
          </AttributeValue>
          <ResourceAttributeDesignator
    AttributeId=”urn:some:path:peptype”
    DataType=”http://www.w3.org/2001/XMLSchema#String”/>
        </ResourceMatch>
      </Resource>
    </Resources>
    </Target>
    <Rule
          RuleId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:rule"
          Effect="Permit">
      <Description>
        User with role "researcher"  from VO "EGEE" can access Resource "CE".
      </Description>
      <Target>
        <Subjects>
           <AnySubject/>
        </Subjects>
        <Resources>
          <AnyResource/>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch
                MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue
                    DataType="http://www.w3.org/2001/XMLSchema#string">SubmitJob</AttributeValue>
               <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                   DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
           <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">researcher</AttributeValue>
        </Apply>
        <SubjectAttributeDesignator
             AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-role"
             DataType="http://www.w3.org/2001/XMLSchema#string"
             Issuer="EGEEAttributeIssuer"/>
      </Condition>
    </Rule>
    <Obligations>
      <Obligation
          ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.UID"
          FulfillOn="Permit">
        <AttributeAssignment
          AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:access-subject"
          DataType="http://www.w3.org/2001/XMLSchema#string">
             &lt;SubjectAttributeDesignator
               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
          </AttributeAssignment>
<!-- This is actual account attribute/name to which it should be mapped -->
          <AttributeAssignment
               AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:poolaccount"
               DataType="http://www.w3.org/2001/XMLSchema#string">
            &lt;PoolAccountDesignator
              AttributeId="http://glite.egee.org/JRA1/Authz/XACML/obligation/poolaccount”
              DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">egee-pool-next-available
         </AttributeValue>
       </AttributeAssignment>
     </Obligation>
   <Obligation
       ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.GID"
       FulfillOn="Permit">
     <AttributeAssignment
          AttributeId="urn:oasis:names:tc:xacml:1.0:scas-policy:subject:subject-group"
         DataType="http://www.w3.org/2001/XMLSchema#string">
      GID-researchers
     </AttributeAssignment>
   </Obligation>
  </Obligations>
</Policy>
```

b) XACML request message example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request
      xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
        access_control-xacml-2.0-context-schema-os.xsd">
  <Subject>
    <Attribute
          AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
          DataType="http://www.w3.org/2001/XMLSchema#string">
       <AttributeValue>Wim Huizinga</AttributeValue>
    </Attribute>
    <Attribute
          AttributeId="urn:oasis:names:tc:xacml:2.0:subject:subject-vo"
          DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>VO-EGEE</AttributeValue>
    </Attribute>
    <Attribute
          AttributeId="urn:oasis:names:tc:xacml:2.0:subject:subject-role"
          DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>researcher</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute
         AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://nikhef.nl/VO-EGEE/CE01</AttributeValue>
    </Attribute>
    <Attribute
         AttributeId="urn:some:path:peptype"
         DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>CE-GT4</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
         DataType="http://www.w3.org/2001/XMLSchema#string">
       <AttributeValue>SubmitJob</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

c) Example Response message with returned obligations containing obligations to assign/map to UID and GID

```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os access_control-xacml-2.0-context-
schema-os.xsd">
  <Result ResourceId=" http://nikhef.nl/VO-EGEE/CE01">
    <Status>
      <StatusCode Value="OK"/>
      <StatusDetail>DecisionID</StatusDetail>
      <StatusMessage>Request is successful</StatusMessage>
    </Status>
    <Decision>Permit</Decision>
    <Obligations>
      <Obligation
          ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.UID"
          FulfillOn="Permit">
        <AttributeAssignment
          AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:access-subject"
          DataType="http://www.w3.org/2001/XMLSchema#string">
             &lt;SubjectAttributeDesignator
               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
          </AttributeAssignment>
<!-- This is actual account attribute/name to which it should be mapped -->
          <AttributeAssignment
               AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:poolaccount"
               DataType="http://www.w3.org/2001/XMLSchema#string">
            &lt;PoolAccountDesignator
              AttributeId="http://glite.egee.org/JRA1/Authz/XACML/obligation/poolaccount”
              DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">egee-pool-next-available
         </AttributeValue>
       </AttributeAssignment>
     </Obligation>
   <Obligation
      ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.GID"
      FulfillOn="Permit">
     <AttributeAssignment
```

```
          AttributeId="urn:oasis:names:tc:xacml:1.0:scas-policy:subject:subject-group"
         DataType="http://www.w3.org/2001/XMLSchema#string">
      GID-researchers
     </AttributeAssignment>
   </Obligation>
  </Obligations>
</Response>
```

d) Response message after transforming by stateful pool accounts manager or UID obligation handler – UID attribute value changed from declarative “egee-pool-next-available” to a specific pool account “egee-pool01”.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os access_control-xacml-2.0-context-
schema-os.xsd">
  <Result ResourceId=" http://nikhef.nl/VO-EGEE/CE01">
    <Status>
      <StatusCode Value="OK"/>
      <StatusDetail>DecisionID</StatusDetail>
      <StatusMessage>Request is successful</StatusMessage>
    </Status>
    <Decision>Permit</Decision>
    <Obligations>
      <Obligation
          ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.UID"
          FulfillOn="Permit">
        <AttributeAssignment
          AttributeId="urn:oasis:names:tc:xacml:1.0:example:attribute:access-subject"
          DataType="http://www.w3.org/2001/XMLSchema#string">
             &lt;SubjectAttributeDesignator
               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
               DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
          </AttributeAssignment>
<!-- This is actual account attribute/name to which it should be mapped -->
          <AttributeAssignment
               AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:poolaccount"
               DataType="http://www.w3.org/2001/XMLSchema#string">
            &lt;PoolAccountDesignator
              AttributeId="http://glite.egee.org/JRA1/Authz/XACML/obligation/poolaccount”
              DataType="http://www.w3.org/2001/XMLSchema#string"/&gt;
         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">egee-pool-next-available
         </AttributeValue>
       </AttributeAssignment>
     </Obligation>
   <Obligation
       ObligationId="urn:oasis:names:tc:xacml:2.0:scas-policy:example007:policy:obligation.GID"
       FulfillOn="Permit">
     <AttributeAssignment
          AttributeId="urn:oasis:names:tc:xacml:1.0:scas-policy:subject:subject-group"
         DataType="http://www.w3.org/2001/XMLSchema#string">
      GID-researchers
     </AttributeAssignment>
   </Obligation>
  </Obligations>
</Response>
```

Appendix E ForCES model in TBS-IP

The XML for the LFB’s of the ForCES based TBS can be found in the ForCES model draft [43].

E.1 Schema of RX messages

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="http://ietf.org/forces/1.0/lfbmodel" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://ietf.org/forces/1.0/lfbmodel:\Projects\Phosphorus\FORCES~1\LFBschemas.xsd"
provides="RxLFB">
    dataTypeDefs>
    <dataTypeDef>
        name>Ports</name>
        synopsis>Values that can be applied to Incoming and Outoging Port for Condition
Forwarding</synopsis>
        atomic>
        <baseType>u8</baseType>
        <rangeRestriction>
            allowedRange max="0" min="9"></allowedRange>
        </rangeRestriction>
        /atomic>
    </dataTypeDef>
    <dataTypeDef>
        name>Connector</name>
        synopsis>A Conditional connector between an incoming port and outgoing port</synopsis>
        struct>
        <element elementID="1">
            name>IncomingPort</name>
            synopsis>The Incoming Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        <element elementID="2">
            name>Condition</name>
            synopsis>The condition upon which from 1 will go to 2</synopsis>
            typeRef>u8</typeRef>
        </element>
        <element elementID="3">
            name>OutgoingPort</name>
            synopsis>The Outgoing Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        /struct>
    </dataTypeDef>
    /dataTypeDefs>
    LFBClassDefs>
    <LFBClassDef LFBClassID="3">
        name>RxLFB</name>
        synopsis>The port Rx LFB for the FE - TVS</synopsis>
        version>1.0</version>
        attributes>
        <attribute elementID="1" access="read-write">
            name>Condition_Fwd</name>
            synopsis>An array with condition forwarding</synopsis>
            typeRef>Connector</typeRef>
        </attribute>
        <attribute elementID="2" access="read-reset">
            name>Packet_Count</name>
            synopsis>The number of packets that are coming into the IXP</synopsis>
            typeRef>uint32</typeRef>
        </attribute>
        /attributes>
    </LFBClassDef>
    /LFBClassDefs>
</LFBLibrary>
```

E.2 Schema of Tx messages

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="http://ietf.org/forces/1.0/lfbmodel" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://ietf.org/forces/1.0/lfbmodel:\Projects\Phosphorus\FORCES~1\LFBschemas.xsd"
provides="TxLFB">
<LFBClassDefs>
    LFBClassDef LFBClassID="4">
    <name>TxLFB</name>
    <synopsis>The port Tx LFB for the FE - TVS</synopsis>
    <version>1.0</version>
    <attributes>
        attribute elementID="1" access="read-reset">
        <name>Packet_Count</name>
        <synopsis>The number of packets that are going out of the IXP</synopsis>
        <typeRef>uint32</typeRef>
        /attribute>
    </attributes>
    /LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

E.3 Schema of TB messages

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="http://ietf.org/forces/1.0/lfbmodel" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://ietf.org/forces/1.0/lfbmodel:\Projects\Phosphorus\FORCES~1\LFBschemas.xsd"
provides="TBLFB">
<dataTypeDefs>
    dataTypeDef>
    <name>Tuple</name>
    <synopsis>A Tuple from the TVS Ticket</synopsis>
<!-- LRI, TokenKey, NewLRI, NewTokenKey, Port1, Port2, IPPacketMask, IPSource, IPDestination,
PortSource, PortDestination, Status -->
    <struct>
        element elementID="1">
        <name>LRI</name>
        <synopsis>Local Reference Identifier</synopsis>
        <typeRef>byte[20]</typeRef>
        /element>
        element elementID="2">
        <name>TokenKey</name>
        <synopsis>A key used for building the encrypted token</synopsis>
        <typeRef>byte[20]</typeRef>
        /element>
        element elementID="3">
        <name>NewLRI</name>
        <synopsis>A Local Reference Identifier for the next domain</synopsis>
        <typeRef>byte[20]</typeRef>
        /element>
        element elementID="4">
        <name>NewTokenKey</name>
        <synopsis> A key used by the next domain</synopsis>
        <typeRef>byte[20]</typeRef>
        /element>
        element elementID="5">
        <name>Port1</name>
        <synopsis>Port 1 Number</synopsis>
        <typeRef>u8</typeRef>
        /element>
        element elementID="6">
        <name>Port2</name>
        <synopsis>Port 2 Number</synopsis>
        <typeRef>u8</typeRef>
        /element>
        element elementID="7">
        <name>IPPacketMask</name>
        <synopsis>IPPacketMask</synopsis>
        <typeRef>byte[20]</typeRef>
        /element>
        element elementID="8">
        <name>IPSource</name>
        <synopsis>IPSource</synopsis>
        <typeRef>byte[4]</typeRef>
        /element>
        element elementID="9">
        <name>IPDestination</name>
        <synopsis>IPDestination</synopsis>
        <typeRef>byte[4]</typeRef>
        /element>
        element elementID="10">
        <name>PortSource</name>
        <synopsis>Port Source</synopsis>
        <typeRef>byte[2]</typeRef>
        /element>
        element elementID="11">
        <name>PortDestination</name>
        <synopsis>Port Destination</synopsis>
        <typeRef>byte[2]</typeRef>
        /element>
        element elementID="12">
        <name>Status</name>
        <synopsis>A Status</synopsis>
        <typeRef>string</typeRef>
        /element>
    </struct>
    /dataTypeDef>
    dataTypeDef>
    <name>ActiveTuples</name>
    <synopsis>An Array with the Active Tuples</synopsis>
    <array type="variable-size">
        typeRef>ActiveTupleCount</typeRef>
    </array>
    /dataTypeDef>
    <dataTypeDef>
        name>Ports</name>
        synopsis>Values that can be applied to Incoming and Outoging Port for a Connector</synopsis>
        atomic>
        <baseType>u8</baseType>
        <rangeRestriction>
            allowedRange max="0" min="255"></allowedRange>
        </rangeRestriction>
        /atomic>
    </dataTypeDef>
    <dataTypeDef>
        name>Connector</name>
        synopsis>A Conditional Connector between an incoming port and outgoing port</synopsis>
        struct>
        <element elementID="1">
            name>IncomingPort</name>
            synopsis>The Incoming Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        <element elementID="2">
            name>Condition</name>
            synopsis>The condition upon which from 1 will go to 2</synopsis>
            typeRef>u8</typeRef>
        </element>
        <element elementID="3">
            name>OutgoingPort</name>
            synopsis>The Outgoing Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        /struct>
    </dataTypeDef>
    <dataTypeDef>
        name>ActiveTupleCount</name>
        synopsis>An Active Tuples with it's counters</synopsis>
        struct>
        <element elementID="1">
            name>ActiveTuplesEntry</name>
            synopsis>An Active Tuples Entry</synopsis>
            typeRef>Tuple</typeRef>
        </element>
        <element elementID="2">
            name>Token_Count</name>
            synopsis>The number of tokens that are coming into the TB</synopsis>
            typeRef>uint32</typeRef>
        </element>
        <element elementID="3">
            name>Bad_Token_Count</name>
            synopsis>The number of bad tokens that have been checked</synopsis>
            typeRef>uint32</typeRef>
        </element>
        /struct>
    </dataTypeDef>
</dataTypeDefs>
<LFBClassDefs>
    LFBClassDef LFBClassID="5">
    <name>TBLfb</name>
    <synopsis>Token Builder LFB</synopsis>
    <version>1.0</version>
    <attributes>
        attribute elementID="1" access="read-write">
        <name>Tuples</name>
        <synopsis>The tuples inside the TB</synopsis>
        <typeRef>ActiveTuples</typeRef>
        /attribute>
        <attribute elementID="2" access="read-write">
            name>Condition_Fwd</name>
            synopsis>An array with condition forwarding</synopsis>
            typeRef>Connector</typeRef>
        </attribute>
        <attribute elementID="3" access="read-reset">
            name>Packet_Count</name>
            synopsis>The number of packets that are coming into the TB module</synopsis>
            typeRef>uint32</typeRef>
        </attribute>
    </attributes>
    /LFBClassDef>
</LFBClassDefs>
</LFBLibrary>
```

E.4 Schema of TS messages

```
<?xml version="1.0" encoding="UTF-8"?>
<LFBLibrary xmlns="http://ietf.org/forces/1.0/lfbmodel" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
```

```
xsi:schemaLocation="http://ietf.org/forces/1.0/lfbmodel:\Projects\Phosphorus\FORCES~1\LFBschemas.xsd"
provides="TSLFB">
    dataTypeDefs>
    <dataTypeDef>
        name>Tuple</name>
        synopsis>A Tuple from the TVS Ticket</synopsis>
        !-- LRI, TokenKey, NewLRI, NewTokenKey, Port1, Port2, IPPacketMask, IPSource, IPDestination,
PortSource, PortDestination, Status -->
        struct>
        <element elementID="1">
            name>LRI</name>
            synopsis>Local Reference Identifier</synopsis>
            typeRef>byte[20]</typeRef>
        </element>
        <element elementID="2">
            name>TokenKey</name>
            synopsis>A key used for building the encrypted token</synopsis>
            typeRef>byte[20]</typeRef>
        </element>
        <element elementID="3">
            name>NewLRI</name>
            synopsis>A Local Reference Identifier for the next domain</synopsis>
            typeRef>byte[20]</typeRef>
        </element>
        <element elementID="4">
            name>NewTokenKey</name>
            synopsis>A key used by the next domain</synopsis>
            typeRef>byte[20]</typeRef>
        </element>
        <element elementID="5">
            name>Port1</name>
            synopsis>Port 1 Number</synopsis>
            typeRef>u8</typeRef>
        </element>
        <element elementID="6">
            name>Port2</name>
            synopsis>Port 2 Number</synopsis>
            typeRef>u8</typeRef>
        </element>
        <element elementID="7">
            name>IPPacketMask</name>
            synopsis>IPPacketMask</synopsis>
            typeRef>byte[20]</typeRef>
        </element>
        <element elementID="8">
            name>IPSource</name>
            synopsis>IPSource</synopsis>
            typeRef>byte[4]</typeRef>
        </element>
        <element elementID="9">
            name>IPDestination</name>
            synopsis>IPDestination</synopsis>
            typeRef>byte[4]</typeRef>
        </element>
        <element elementID="10">
            name>PortSource</name>
            synopsis>Port Source</synopsis>
            typeRef>byte[2]</typeRef>
        </element>
        <element elementID="11">
            name>PortDestination</name>
            synopsis>Port Destination</synopsis>
            typeRef>byte[2]</typeRef>
        </element>
        <element elementID="12">
            name>Status</name>
            synopsis>A Status</synopsis>
            typeRef>string</typeRef>
        </element>
        /struct>
    </dataTypeDef>
    <dataTypeDef>
        name>ActiveTuples</name>
        synopsis>An Array with the Active Tuples and their counters</synopsis>
        array type="variable-size">
        <typeRef>ActiveTupleCount</typeRef>
        /array>
    </dataTypeDef>
    <dataTypeDef>
        name>ActiveTupleCount</name>
        synopsis>An Active Tuples with it's counters</synopsis>
        struct>
        <element elementID="1">
            name>ActiveTuplesEntry</name>
            synopsis>An Active Tuples Entry</synopsis>
            typeRef>Tuple</typeRef>
        </element>
        <element elementID="2">
            name>Token_Count</name>
            synopsis>The number of tokens that are coming into the TS</synopsis>
            typeRef>uint32</typeRef>
        </element>
        <element elementID="3">
            name>Bad_Token_Count</name>
            synopsis>The number of bad tokens that have been checked</synopsis>
            typeRef>uint32</typeRef>
        </element>
        /struct>
    </dataTypeDef>
    <dataTypeDef>
        name>Ports</name>
        synopsis>Values that can be applied to Incoming and Outoging Port for a Connector</synopsis>
        atomic>
        <baseType>u8</baseType>
        <rangeRestriction>
            allowedRange max="0" min="255"></allowedRange>
        </rangeRestriction>
        /atomic>
    </dataTypeDef>
    <dataTypeDef>
        name>Connector</name>
        synopsis>A Conditional Connector between an incoming port and outgoing port</synopsis>
        struct>
        <element elementID="1">
            name>IncomingPort</name>
            synopsis>The Incoming Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        <element elementID="2">
            name>Condition</name>
            synopsis>The condition upon which from 1 will go to 2</synopsis>
            typeRef>u8</typeRef>
        </element>
        <element elementID="3">
            name>OutgoingPort</name>
            synopsis>The Outgoing Port/MicroEngine</synopsis>
            typeRef>Ports</typeRef>
        </element>
        /struct>
    </dataTypeDef>
    /dataTypeDefs>
    LFBClassDefs>
    <LFBClassDef LFBClassID="5">
        name>TSLfb</name>
        synopsis>Token Switch LFB</synopsis>
        version>1.0</version>
        attributes>
        <attribute elementID="1" access="read-write">
```

```
            name>Tuples</name>
            synopsis>The tuples inside the TB</synopsis>
            typeRef>ActiveTuples</typeRef>
        </attribute>
        <attribute elementID="2" access="read-write">
            name>Condition_Fwd</name>
            synopsis>An array with condition forwarding</synopsis>
            typeRef>Connector</typeRef>
        </attribute>
        /attributes>
        events baseID="1">
        <event eventID="1">
            name>BadCountSurpassed</name>
            synopsis>If the Bad Count is surpassed the LFB will send a message to the CE</synopsis>
            eventTarget>
            <eventField>Tuples</eventField>
            <eventSubscript>_TupleEntry_</eventSubscript>
            <eventField>BadTokenCount</eventField>
            /eventTarget>
            eventCreated>
            <eventGreaterThan>255</eventGreaterThan>
            /eventCreated>
            eventReports>
            <eventReport>
                eventField>Tuples</eventField>
                eventSubscript>_TupleEntry_</eventSubscript>
                eventField>ActiveTuplesArray</eventField>
                eventField>LRI</eventField>
            </eventReport>
            /eventReports>
        </event>
        /events>
    </LFBClassDef>
    /LFBClassDefs>
</LFBLibrary>
```