OMA Service Environment

Approved Version 1.0.4 – 01 Feb 2007

Open Mobile Alliance

OMA-AD-Service-Environment-V1_0_4-20070201-A

Contents

Figures

Tables

1. Scope

This document describes the OMA Service Environment (OSE), which is a flexible and extensible architecture that offers support to a diverse group of application developers and Service Providers.

The primary intention of the OSE is to promote common architectural principles, across the whole of OMA, for how OMA Enablers are specified and how they interact with one another whilst ensuring architecture integrity, scalability and interoperability, all of which strive to reduce Architecture *silo* design and hence reduce integration and deployment complexities.

This document includes the following information:

- A high-level description of the OMA Service Environment including its concepts and entities;
- A migration path between existing and future enabler activities within the OMA;
- A description of how OMA enablers interact with one another, for example, to support the creation and delivery of widely accessible coherent end-user services.

2. References

2.1 Normative References

[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

2.2 Informative References

[ARCH-INVEN]	"Inventory of Architectures and Services", Inventory-of-Architectures-and-Services-V1_0, Open Mobile Alliance™, URL: http://www.openmobilealliance.org/
[ARCH-REQ]	"OMA Architecture Requirements, Open Mobile Alliance™, OMA-RD-Architecture-V1_0, URL: http://www.openmobilealliance.org/
[GSM 01.04]	"Abbreviations and acronyms", European Telecommunications Standards Institute, Technical Report GSM 01.04. URL: http://www.3gpp.org
[ITU-T I.112]	"Vocabulary of terms for ISDNs", International Telecommunication Union, ITU-T Recommendation I.113, URL: http://www.itu.org/
[OMA-DICT]	"Dictionary for OMA Specifications", OMA-ORG-Dictionary-V2_5, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/
[RFC 2828]	"Internet Security Glossary", RFC 2828, URL:http://www.ietf.org/rfc/rfc2828.txt
[TMF]	"Product Lifecycle Management with NGOSS", TMFC2051 PLM Catalyst Overview-final, URL:http://www.tmforum.org/
[X.800]	"Security architecture for Open Systems Interconnection for CCITT applications", ITU, Recommendation X.800, URL: http://www.itu.int/rec/T-REC-X.800/en

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendices are informative, unless they are explicitly indicated to be normative.

All figures in this document are informative.

3.2 Definitions

For the purposes of this document, the terms and definitions given in [OMA-DICT] apply and the following also apply:

Application	An implementation of a related set of functions that perform useful work, often enabling one or more services. It may consist of software and/or hardware elements.
Authentication	The process of verifying an identity claimed by or for a system entity. [RFC 2828]
Authorization, **Authorize**	(1.) An "authorization" is a right or a permission that is granted to a system entity to access a system resource. (2.) An "authorization process" is a procedure for granting such rights. (3.) To "authorize" means to grant such a right or permission. [RFC 2828]
Delegate	A delegate is a designated system or resource that performs specified tasks or functions on behalf of (one or more) other systems. To *delegate* is to designate a system or resource that performs specified tasks or functions on behalf of (one or more) other systems.
Interface	The common boundary between two associated systems. [GSM 01.04, ITU-T I.112]
Intrinsic function	Intrinsic functions are those functions that are essential in fulfilling the intended task of the specified enabler
Logical Architecture	Incorporates the detailed architecture diagram (with interfaces), elements and interface specifications. This architecture is used to derive detailed architecture based upon which an implementation can be made.
OMA Service Environment	A logical architecture that provides a common structure and rule set for specifying enablers.
Non-intrinsic function	Non-intrinsic functions are those functions that are not essential in fulfilling the intended task of the specified enabler
Parameter P	Parameter P is the set of parameters that must be added to requests through the I0 interface of an enabler in order to satisfy existing policies to be enforced when exposing this enabler.
Policy	A policy is uniquely represented by a logical combination of conditions and actions.
Policy Enforcement	The process of executing actions, which may be performed as a consequence of the output of the policy evaluation process or during the policy evaluation process.
Policy Evaluation	The process of evaluating the policy conditions and executing the associated policy actions up to the point that the end of the policy is reached.
Policy Processing	Policy evaluation or policy evaluation and enforcement
Request	An articulation of the need to access a resource or to invoke a function. A request may include zero, one or more facts.
Requestor	Any entity that issues a request to a resource.
Resource	Any component, enabler, function or application that can receive and process requests.
Risk	An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particularly harmful result.
Service Enabler	A technology intended for use in the development, deployment or operation of a Service; defined in a specification, or group of specifications, published as a package by OMA.

Threat	A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm.

3.3 Abbreviations

For the purposes of this document, the abbreviations given in [OMA-DICT] apply and the following also apply:

API	Application Programming Interface
ETOM	extended Telecommunications Operations Map
JCP	Java Community Process
OASIS	Organization for the Advancement of Structured Information Standards
OMA	Open Mobile Alliance
OSE	OMA Service Environment
PE	Policy Enforcer
PEEM	Policy Evaluation, Enforcement and Management
PoC	Push to talk over Cellular
SID	Shared Information/Data model
SNMP	Simple Network Management Protocol
TMF	TeleManagement Forum

4. Introduction

4.1 General

The OMA specifies enablers, which provide standardized components to create an environment in which services may be developed and deployed. The OMA enablers, the decomposition into these components and the interactions between them comprise the OSE.

The primary intention of the OSE is to address the issues as described in Section 4.3 and to satisfy the OMA Architecture requirements [ARCH-REQ] which focuses on:

- High-level functional and system requirements that describe the need for architecture integrity, scalability, interoperability and the elimination of *silo* designs and hence the reduction of integration and deployment complexities and the importance of security, usability and privacy;
- Overall system element requirements that describe general enabler and interface requirements.

In general, the OSE addresses the issues as described in Section 4.3 by simplifying:

- The controlled exposure of resources to internal and Third Party application developers, in order for them to create and run compelling new services;
- The integration and management of resources;
- The evolution of OMA's current *silo-like* conglomerate architecture to an integrated unified and well coordinated OMA service enabler environment.

The use of the OSE will lead to:

- Rapid development and deployment of new and innovative applications;
- Reuse of OMA enablers and the reduction of *silos*;
- Opening up service creation to Third Parties while protecting the Service Providers assets;
- Enabling the use of varied business models for services;
- Broadening of the developer pool;
- Making automated management of business relationships possible;
- Development of an evolution path for an integrated and unified service enabler environment.

The remainder of this document will further elaborate on these topics.

4.2 Targeted audience and document intention

This document is targeted at OMA members, other specification-defining organizations, and companies who want to make use of OMA-defined specifications.

The intention of the OSE is to:

- Provide guidance to specification writers when creating new or evolving existing OMA enablers;
- Assist in understanding the relations and interactions between OMA enablers as well as non-OMA resources, specifically to encourage and simplify reuse.

4.3 Motivation

4.3.1 Existing service development and integration

Service architectures specified today are created by standards bodies and are targeted at a particular service. When individual enablers are defined without the benefit of an overall architecture, each enabler will be forced to define all functions required to fulfil its requirements. This monolithic approach to enablers creates a number of issues for the Service Provider:

- Integration and deployment of services is complicated and expensive;
- High implementation efforts for applications wanting to use several capabilities;
- There is no common integration of the different services from the point of view of the end-user (e.g. no common group management or user profile across multiple services).

The term *silo* has become popular in this context as it highlights the fact that the implementation of the service has been done by integrating different components vertically and per-service. Implementation and integration work done for one service cannot be reused in others due to the lack of standards.

The *silo* nature of both standards and products results in a number of problems that raise costs and slow down deployment for new services. From a Service Provider's perspective:

- Integration with underlying network infrastructure must be repeated for each deployment, which results in duplication of integration work;
- Many functions and their associated data are duplicated with the introduction of new services, e.g. each service implementation tends to have its own subscriber database, or its own way of authenticating end-users or accounting for service usage;
- Sharing of, for example, the preferred notification method (email, SMS or voice call) across services requires costly integration activities.

Another problem of the *silo* architecture is that each service comes typically with its own management facilities, and the way the service is actually deployed in the network is also different. In addition the *silo* architecture of services also requires detailed knowledge about the network to integrate service implementation with the underlying network infrastructure, or with end-user equipment, e.g. terminals. Some components, such as user profiles need to be developed again for each service and cannot be reused for other services. The result is non-satisfactory time-to-market as well as high costs and inconsistent user interfaces across multiple services.

4.3.2 End-user perception

From an end user perspective, the independent deployment of services leads to inconsistent user experience when using different services offered by a single provider or even when using the same service across different environments (e.g. caused whilst end-user is roaming). From the end-user perspective the inconsistency in user experience arises because of:

- Inconsistent reuse of user information, preferences, privacy settings, etc;
- Lack of service continuity caused by user mobility and service mobility;End-users inability to choose how services are accessed and used;
- Limitations in the end-user's perception of their relationship and interaction with other actors (e.g. mobile operators and enterprises), and the roles that each actor fulfils, within the user mobility and service mobility eco-system.

4.3.3 OMA enablers and enabler reuse

The main role of OMA is the specification of OMA enablers, which provide for a number of benefits:

- Enablers provide interoperable components that enable the interaction between different components and applications developed by different providers (e.g. device and network suppliers, information technology companies and Content and Service Providers);
- The specification of enablers reduces deployment efforts and allows the same applications to operate across a wide variety of environments in a consistent manner;
- The specification of enablers also allows for reuse, so that commonly used functions can be provided for by standard components, instead of recreating those same functions in each application.

The latter point emphasises the need to identify potential areas of overlap, especially where OMA provides more than one way of providing the same capability. This is true within a particular area (e.g. location or instant messaging) where there previously existed more than one organization that targeted the same standardisation effort, but also across areas where often the same capabilities are needed, but are provided in different ways.

An integral part in the development of the OSE is to promote the reuse of common functions that may be used by other OMA enablers and non-OMA resources, and to create new OMA enablers that provide those common functions.

In addition, the OSE encourages the identification of gaps between existing standards by analysing different standards (see [ARCH-INVEN]), and if a gap is detected and its associated function is identified as benefiting from standardization, then this gap is a potential candidate for a new OMA enabler.

4.4 OMA *silo* architectures

OMA produces open specifications to create building blocks to provide services to end-users or to maintain or enhance the environment in which services are provided. As described in Section 4.3 these specifications have been developed, in most cases, without a concern for how they interact with each other, nor do they seek to provide unified and consistent structure by, for example, identifying potential areas of overlap and avoiding duplication. For a detailed architecture view of OMA *silo* architectures, which must be avoided in future OMA enabler design and specification, refer to [ARCH-INVEN].

5. The OSE Architecture

5.1 Architecture requirements and principles

The OSE architecture is developed to satisfy the OMA Architecture Requirements [ARCH-REQ]. The Architecture requirements document [ARCH-REQ] describes both functional and system requirements that need to be satisfied by the OSE. The Architecture requirements document [ARCH-REQ] also implies the need for a set of interfaces. These interfaces and the associated OMA architecture requirements are described in Appendix B.

Additionally, the OSE focuses on several key concepts that address the issues as described in Section 4.3.

The OSE architecture can be realized using the specifications, as defined by, for example, Parlay OASIS, JCP and Liberty Alliance. The key principles of the OSE are described in the following sections.

5.1.1 Intrinsic functionality

Intrinsic functions are those functions that are essential in fulfilling the intended task of the specified enabler. For example, the Position Calculation function is Intrinsic to Secure User Plane Location; Authentication is intrinsic to Single Sign On; Encryption is an intrinsic function of Digital Rights Management.

Non-Intrinsic functions are those functions that are not essential in fulfilling the intended task of the specified enabler. For example, Authentication is a non-intrinsic function to Data Synchronisation; Encryption is a non-intrinsic function of Device Management.Any requirements or features that are not intrinsic to an enabler should not be specified within the enabler's specification. An enabler's specification should only specify the intrinsic functionality required to fulfil its actual function.

However, the classification of intrinsic and non-intrinsic functions is relative to its usage by another enabler (see previous example for Encryption).

The classification of intrinsic and non-intrinsic is subjective and needs to be done on a per enabler basis.

5.1.2 Delegation and reuse of enablers

Enabler specifications should reuse existing specifications where possible. This approach includes the reuse of existing OMA enabler specifications whenever possible (e.g. reuse of presence and group management enablers by the PoC enabler). Enabler specifications must specify how to interface to (i.e. invoke) their functions.

As a result of enabler specifications reusing other enabler specifications, the vertical *silo* problem can be reduced. The integration of new applications and enablers into an OSE domain can be simplified. Examples of OSE domain include: an SP domain, a terminal domain, or an enterprise domain. Enabler implementations may reuse other enablers located in either the same OSE domain or across different OSE domains.

An enabler implementation can invoke any standardized function, such as authentication or group management, that it needs to satisfy its intrinsic functions defined in its specifications.

5.1.3 Protection of enablers and resources

In order to protect the underlying resources in an OSE domain from unauthorized requests and to manage the use of these requests it is important that the OSE enables the exposure of OMA enablers, other functions, resources and applications to each other in a controlled manner. It is also important that the OSE architecture manages the procedures applied to enablers and applications that reside either in the same environment or across different environments.

5.1.4 Extensibility

In any OSE domain, implementations of the OMA enablers expose standard interfaces for application and enabler use. These enabler implementations connect to the actual resources present in the OSE domain. Through this abstraction, it is possible to add or modify the underlying resources without affecting the interface exposed by the enabler implementations (and therefore without affecting the applications), something that is especially important when using multiple vendors, supporting different network technologies or relying on different providers.

New enablers can be introduced into any OSE domain by developing an enabler implementation that may connect to an underlying resource in that domain.

The enabler's interfaces are offered by the enabler implementations for use by applications or other enabler implementations. The interfaces follow the OMA specifications and they are technology specific realizations of the specified interfaces (e.g. web services, Java).

The enabler's interface(s) can be registered with the (proposed) discovery enabler to allow applications to dynamically bind to the destination enabler.

One way of controlling access to enablers is to use policies. Policies can be loaded dynamically for policy evaluation and enforcement to protect the enabler.

When required, Policy definitions may help in extensibility by using the delegation mechanism.

Life cycle management interfaces are expected to provide support for upgrade of enablers when new releases are installed and deployed.

5.2 Architectural Model

Section 5 defines the OSE Architecture which is the set of architecture elements and the relationships between these elements. The architecture elements are defined in the subsections of this section. The relationships are the *interface categories* as defined in Section 5.3.

Figure 1 illustrates the architecture elements of both the Service Provider and terminal domains of the OSE. This view focuses on identifying the different elements present in the OSE. The figure is not meant to be indicative of any particular deployment model.

The OSE Architecture does not specify where architectural elements (e.g. applications, enablers, etc.) reside. For example, the architectural elements may reside in a Mobile Operator's network, or on mobile terminals.

Thus, throughout this document, the OSE architecture also applies to a user terminal.

NOTE to the Reader: Further details about the OSE and the terminal will be provided in future releases of the OSE.

The OSE Architecture does not mandate the deployment of a Policy Enforcer implementation or of any enabler implementation in any domain. When an enabler or Policy Enforcer implementation is deployed, the OSE architecture does not mandate a specific deployment model choice. This allows flexibility in how OMA enablers and the Policy Enforcer function are implemented and deployed.

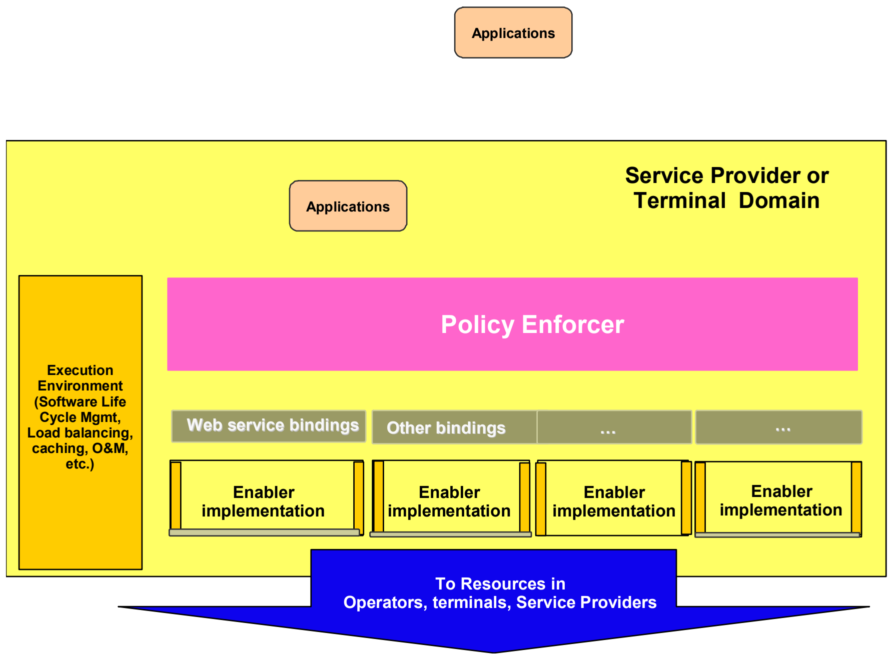
Applications
Applications
Service Provider or Terminal Domain
Execution Environment (Software Life Cycle Mgmt, Load balancing, caching, O&M, etc.)
Policy Enforcer
Web service bindings
Other bindings
...
...
Enabler implementation
Enabler implementation
Enabler implementation
Enabler implementation
To Resources in Operators, terminals, Service Providers

Figure 1 – OSE architecture elements.

Each architecture element is described in the subsequent sections.

5.2.1 Enabler

The enabler (or its long form *Service Enabler*) architecture element is pervasive in OMA because enablers are the primary products of OMA (e.g. Enabler Releases and Enabler Packages). An enabler should specify one or more public interfaces.

Examples of OMA enablers include Location or Device Management.

The term enabler is formally defined in [OMA-DICT] but is copied here for the convenience of the reader:

Enabler *- A technology intended for use in the development, deployment or operation of a Service; defined in a specification, or group of specifications, published as a package by OMA.*

5.2.2 Enabler implementation

Although specifications created by OMA are technology-agnostic regarding their implementation, the reality is that enablers will be implemented in real deployments of service environments. Consequently, this document defines *Enabler Implementations* as an element in the OSE and it literally represents an *implementation of an enabler,* e.g. either in a Service Provider domain or in a terminal domain. An enabler implementation can be viewed as a *template* that represents an implementation of any enabler (e.g. MMS) as defined by OMA. When an enabler specifies multiple entities (e.g. client and server, multiple clients or multiple servers) and their interactions, each of these entities can be implemented as separate enabler implementations (e.g. client enabler implementation and server enabler implementation).

The OSE makes no restrictions on how enabler specifications are implemented.

Enabler implementations provide standardized functions. The enabler implementation may amalgamate, abstract and/or repackage a resource, and present its functions through an interface after binding to a particular syntax.

Enabler implementations expose life cycle management interfaces (e.g. start, stop, trace, etc) that allow the domain to use infrastructure capabilities to manage the enabler's components.

OMA defines many enablers such as location and device management. In addition, other functions (e.g. authentication, access control, discovery and directories) may be provided either through enabler implementations, infrastructure features or applications (e.g. Third Party management and transaction management) available in the environment.

Enabler implementations may be invoked by applications or other enabler implementations. OMA enablers may be defined for usage in callable mode, proxy mode, both or in none of these modes. They are all represented in the OSE as enabler implementations (see Figure 1). Depending on their role or deployment model they will present an interface and be used as proxies or callable enablers.

The enabler implementations process the messages as defined by the enabler specification. The binding elements provide the specific syntax to express these messages in the selected format such as web services, Java or .Net.

5.2.3 Interfaces

The term *Interface* is formally defined in [OMA-DICT] but is copied here for the convenience of the reader:

Interface: *The common boundary between two associated systems (source:* [GSM 01.04, ITU-T I.112]).

This document defines several generic interfaces for the OSE. See “Section 5.3” for more information about these interfaces.

Enabler specifications typically define interfaces to:

- Invoke the intrinsic functions of the enabler specification in an interoperable manner;
- Support interoperability between entities of an enabler;
- Allow the ability to provide life-cycle management of enablers.

However, as a fundamental principle of OMA, enabler specifications do not specify technology-specific Application Program Interfaces (API). The OSE does not specify any APIs.

NOTE: The OSE does not specify any *Reference Point*s (see [OMA-DICT] for a definition of Reference Point).

5.2.4 Enabler interface bindings

Interfaces must be specified in a language neutral manner. However, specifications may also define language specific bindings for the interfaces. Enabler interface bindings provide the specific formats (i.e. syntax and protocols used to access enablers using particular programming languages (e.g. Java or C) or network protocols (e.g. web services).

5.2.5 Resources

A *Resource* in this document is an architecture element that represents a capability in a Service Provider’s domain or terminal domain. In the OSE, an enabler implementation may directly invoke or access a resource.

5.2.6 Applications

The term *Application* is formally defined in [OMA-DICT] but is copied here for the convenience of the reader:

Application*: An implementation of a related set of functions that perform useful work, often enabling one or more services. It may consist of software and/or hardware elements.*

Applications are identified as an element in the OSE because they are a primary means for initiating and consuming an enabler. For example an application may directly invoke an enabler implementation to deliver a service.

Applications may be located anywhere in a service environment including a mobile terminal.

5.2.7 Execution Environment

A full service lifecycle model for services has been defined by the TeleManagement Forum [TMF], and mapped to the eTOM (extended Telecommunications Operations Map). This mapping is defined in an abstract way, which can be adapted to any deployment environment. As an example of an existing specification developed by another open standards group that may meet OMA requirements, OMA should re-use this model.

NOTE: The following is a simplified model that forms a framework for the detailed description of the life-cycle model, which is achieved by mapping the high-level model onto the eTOM [TMF].

The high-level model of the service life cycle contains the following operations/phases:

- Develop;
- Sell;
- Provide;
- Bill;
- Service;
- Report;
- Modify/Exit.

Within the scope of OSE, the *Execution Environment* provides support for software life-cycle management functions. Such functions may be used during the service life-cycle phases defined by [TMF].

The *Execution Environment* is an element in the OSE. This execution environment or platform logically encompasses various functions such as process monitoring, software life cycle management, system support (e.g. thread management, load balancing and caching), operation, management and administration that allow the OSE domain to control enablers. The functions within the Execution Environment may not be directly exposed to applications, however these functions may be directly invoked by enabler implementations. In addition, resources can rely on these functions and may assume that the functionality of the Execution Environment is available. Software life cycle management includes a set of functions of the Service Provider Execution Environment and can be implemented as a separate enabler, or it may be distributed over several enablers.

Then, in the OSE domain, certain software life-cycle management functions are needed to provide basic support to the enabler implementations.

The software life-cycle management functions include but are not limited to:

- Creation;
- Software deployment;
- Software Management:
    - Process Activation & deactivation (e.g. actuation);
    - Dependency management;
    - Upgrade;
    - Removal;
    - Fault management (e.g. logging and SNMP traps);
    - Performance management (e.g. measuring).

For further information on TMF and mapping to the eTOM and the SID (Shared Information/Data model) of the TMF, see [TMF].

5.2.8 Policy Enforcer

The Policy Enforcer (PE) is an OSE architectural element that provides a policy-based management mechanism to protect resources from unauthorized requests and to manage the use of these requests for instance through appropriate charging, logging and evaluation and enforcement of user privacy or preferences. Please refer to Section 5.2 for the deployment aspect of the OSE architecture.

The Policy Enforcer function allows the domain owner to extract and separate their policy rules from architectural elements. The OSE architecture does not describe how the PE is realized. The PE may be realized in several ways, one of which is the PEEM enabler.

The OSE architecture also manages the procedures applied between enablers and applications that reside either in the same environment or across different environments.

5.3 Interfaces of the OSE

Figure 2 illustrates the interface categories of the OSE architecture.

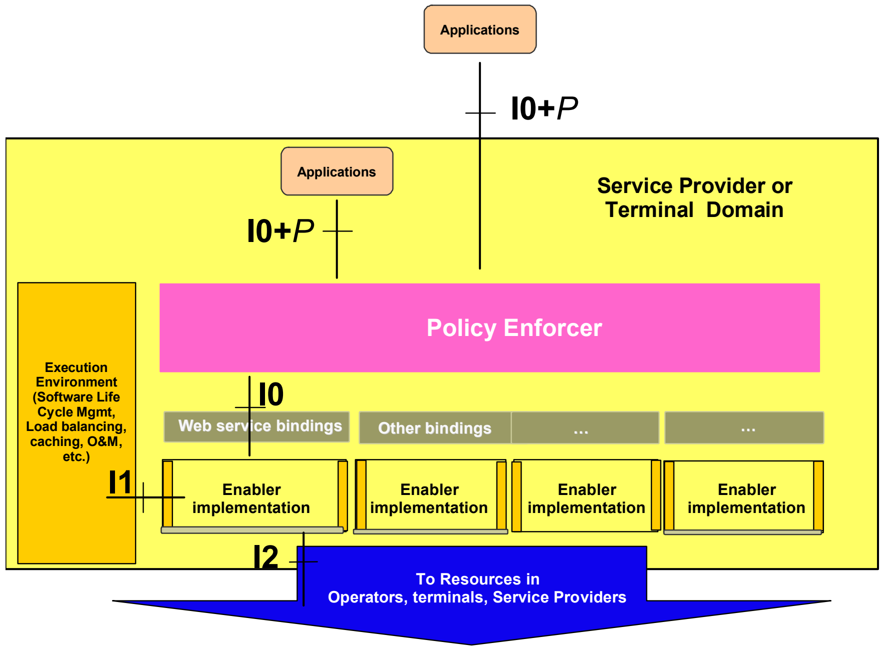

Figure 2 – Classification of interfaces in OSE

Table 1 contains a list of the OSE interface categories including their definition and additional comments.

The interfaces defined in this section are abstract interface categories. For example, the *I0* interface represents the categories of interfaces that enabler specifications (e.g. the OMA Location enabler) define.

Table 1: Interface Categories of the OSE Architecture

Interface category	Definition	Comments
I0	I0 is the category of interface to an enabler's intrinsic functions. I0 interfaces are exposed to applications and enablers when the policies that are to be enforced do not require any additional parameters or when no policy is associated to the request to this enabler. I0 interfaces are specified by OMA (see note 1).	I0 may encompass interfaces to what in some areas are called “service building blocks” like location and messaging, as well as to traditional “business support functions” like subscriber management. The category of I0 interfaces includes asynchronous events and methods to register/subscribe listeners to these events. (See note 2)
I0+P	I0+P is the category of interfaces that combines I0 and P as required to satisfy existing policies that are to be enforced when exposing the I0 interface of the enabler. (See the definition of Parameter P for more information.). I0+P are exposed to applications and enablers when the policies that are to be enforced require additional parameters.	The Policy Enforcer may require additional parameters (*P*) that must be provided along with the request to the enabler’s interface (I0), based on policies specified by any principal who is authorized to do so; i.e. typically the owner or administrator of the OSE domain where the enabler is located. (See note 3)
I1	I1 is the category of interfaces between enablers and the Execution Environment (e.g. software life cycle management process and monitoring etc.). The I1 interfaces may be specified by OMA (see note 1).	
I2	I2 is the category of interfaces used by enablers to describe how to invoke an underlying resource's function. Such interfaces are not defined by OMA.	I2 may encompass interfaces to underlying networks (i.e. mobile operator’s network) as well as to backend resources (i.e. BSS, O&M) (See note 2).

NOTE 1: A new interface can be specified by OMA or OMA can make reference to an existing interface.

NOTE 2: Further elaboration on I0 and I2 interfaces may be provided in future versions of the OSE.

NOTE 3: See Section 5.4 “Applying the OSE Architecture” for a detailed explanation of implications of Policy management on enabler interfaces.

5.4 Applying the OSE architecture

5.4.1 Controlled exposure of enablers and resources

If required by the domain owner, a Policy Enforcer enabler implementation provides a consistent and possibly centralized management mechanism to facilitate controlled access to enablers and resources exposed by the domain. The Policy Enforcer provides a mechanism for domain owners to evaluate and enforce policies for, e.g. security, access control, privacy, or charging, on any request to a domain’s resource (see Figure 3).

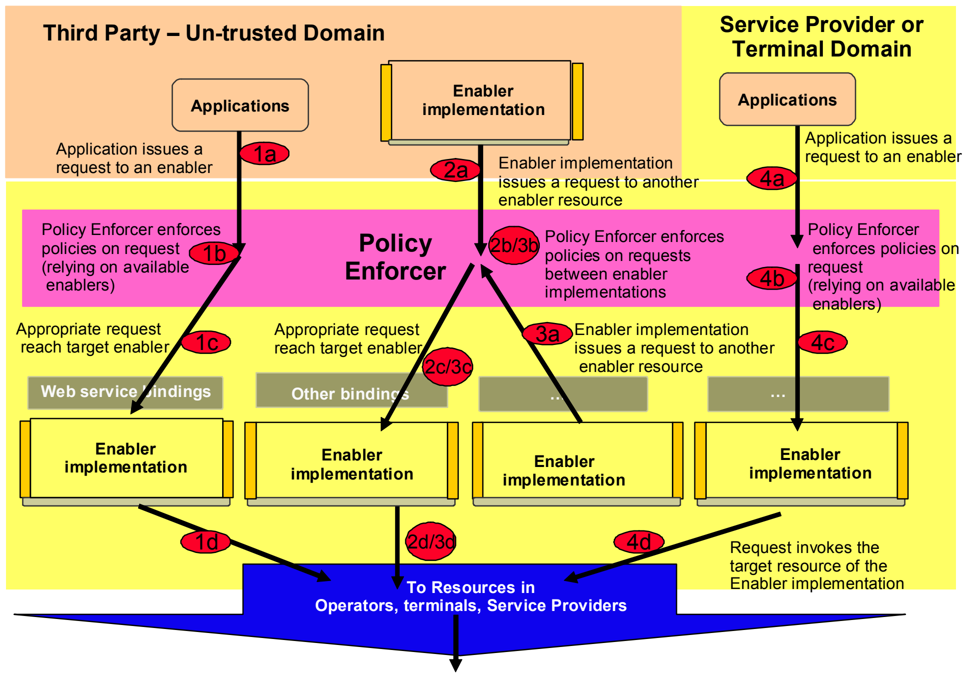
Third Party – Un-trusted Domain
Service Provider or Terminal Domain
Applications
Enabler implementation
Applications
Application issues a request to an enabler
1a
2a
Enabler implementation issues a request to another enabler resource
4a
Application issues a request to an enabler
Policy Enforcer enforces policies on request (relying on available enablers)
1b
Policy Enforcer
2b/3b
Policy Enforcer enforces policies on requests between enabler implementations
4b
Policy Enforcer enforces policies on request (relying on available enablers)
Appropriate request reach target enabler
1c
Appropriate request reach target enabler
2c/3c
3a
Enabler implementation issues a request to another enabler resource
4c
Web service bindings
Other bindings
...
...
Enabler implementation
Enabler implementation
Enabler implementation
Enabler implementation
1d
2d/3d
4d
Request invokes the target resource of the Enabler implementation
To Resources in Operators, terminals, Service Providers

Figure 3 - OSE Flows

The Policy Enforcer architectural element supports evaluation and enforcement of policies that are able to invoke, for example, authentication and authorization, when an enabler implementation is able to delegate (or reuse), for example, authentication and authorization.

The Policy Enforcer may use enablers to evaluate and enforce the policies that have been specified for the domain and/or the target enabler. The Policy Enforcer may also be used to compose enablers into higher-level functions. The interfaces to these higher-level functions, although based on composition of defined OMA enablers, have not been defined in OMA and may or may not be modelled as IO+P.

The policies evaluated and enforced by the Policy Enforcer determine the types of messages exchanged from and to the requester and to and from the target and delegated enablers. Therefore, policies can be used to dictate the interface exposed to requesters as well as modify the exposed bindings. An I0 interface may be differently exposed via domain owner policies, but may or may not be be modelled as IO+P. Such complex transformations may include addition or subtraction of functionality, but also more radical transformations such as bindings transformations.

The Policy Enforcer can be invoked by any other authorized (as determined by the policies associated to the Policy Management) element of the OSE to evaluate and enforce policies.

The Policy Enforcer applies the same rigid procedures for enablers and applications that reside either in the same environment or across different environments. This is achieved by having the Policy Enforcer process all requests to and from the enabler implementations and evaluate and enforce the appropriate policies.The specific logic used to handle (e.g. route or intercept) such requests is also part of the Policy Enforcer function, regardless of implementation.

The domain that provides a resource may set policies. These Policies may also be combined with other Policies derived from preferences or rules set up by end-users or from the terms and conditions (Service Level Agreements) agreed for third parties to use a resource. The domain may also evaluate and enforce additional policies on behalf of other parties.

The domain will only associate policies (and the resulting evaluation and possibly also the enforcement) to an enabler implementation that is able to delegate functionality.

Components providing the policy enforcer function are not required to be deployed in the OSE when deployments do not need policies to be applied to exposed enabler implementations. When an enabler is able to delegate functions such as authentication, the domain can supply policies evaluated and enforced by the Policy Enforcer to perform these functions.

A request originating from an application or an enabler implementation may arrive to the Policy Enforcer architectural element through a variety of mechanisms and may be processed in a variety of ways.

5.4.2 Using the exposed resources

Figure 4 illustrates the steps of determining which interfaces are associated to a target enabler. Steps 1a/1b describe two alternative steps at application development time. Step 1c is an alternative discovery process that can take place at execution. After the establishment of a relationship, a third party can discover the resources exposed by the domain. This may be achieved through the use of a discovery service or enabler. It is also possible that the interfaces of a resource are communicated with other exchanges between the domain and the application developer when developing the application.

After the applications bind to the enabler interfaces the Policy Enforcer processes the exchanges to control third party access to the enablers. As an architectural element, the Policy Enforcer controls any exchanges. However, there may be cases when the policy to be applied may be a *zero* policy whereby the Policy Enforcer does not have to process the request. When a domain owner chooses not to apply policies on requests, since the Policy Enforcer does not process any requests (e.g. all policies are "zero" policies), hence the domain owner may choose not to deploy the Policy Enforcer.

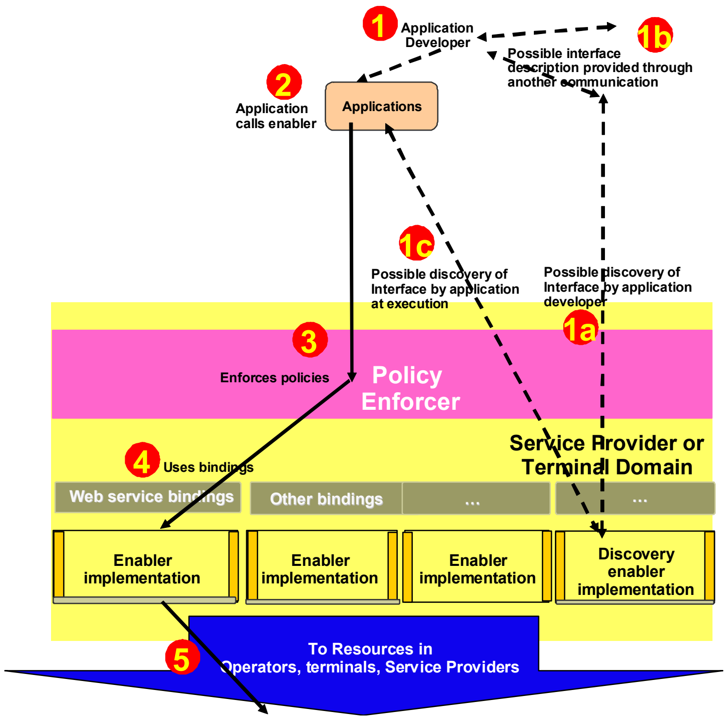
1
Application Developer
1b
Possible interface description provided through another communication
2
Application calls enabler
Applications
1c
Possible discovery of Interface by application at execution
Possible discovery of Interface by application developer
1a
3
Enforces policies
Policy Enforcer
Service Provider or Terminal Domain
4
Uses bindings
Web service bindings
Other bindings
...
...
Enabler implementation
Enabler implementation
Enabler implementation
Discovery enabler implementation
5
To Resources in Operators, terminals, Service Providers

Figure 4 - Third Party engagement steps

5.4.3 Implications of policy management on enabler interfaces

5.4.3.1 Interfaces towards Third Parties

The Policy Enforcer provides controlled access to enablers and resources exposed by the domain. The enabler implementations process messages as defined by the enabler specification. The binding elements provide the specific syntax to express these messages in the selected format such as web services, Java or .Net.

5.4.3.2 Interface I0 and I0+*P*

Appropriate design of the enabler specifications should allow the separation between the domain owner-defined parameters (*P*) and the parameters core to the enabler interface (I0).

This distinction between interface I0 and I0+*P* allows the enabler developer to implement the enabler interface (I0) specification, which requests only the parameters associated to the enabler core functionality.

However, in general, interface I0 and I0+*P* could be considered as being different interfaces. Therefore, if an enabler has been designed to be reused by other enablers or applications, the enabler interface (I0) should only support the procedures and parameters needed to invoke the enabler's core functions, for example, location parameters in the location enabler.

When the domain imposes policies, for example, when requiring authentication, authorization or charging, the request towards the enabler must deliver the necessary information. However, considering that the enabler interface (I0) is only capable of supporting the enabler's core procedures and parameters (e.g. location parameters) it is necessary for the Policy Enforcer to utilise interface I0+*P* and process the authentication, authorization or charging parameters to ensure that the domain's imposed policies are satisfied.

An enabler developer implements the enabler interface (I0) that requests only the parameters in interface (I0). Domains are then able to request additional parameters (e.g. charging tokens, identity credentials), as needed by their policies, in order to correctly access the resource. The I0 with these additional parameters constitutes I0+*P*. This however does not affect the application developer and application portability.

5.4.4 Deployment options

Policy Enforcer is an architectural element of the OSE. The Policy Enforcer may be realised by the OMA PEEM enabler.

Deployment options for the Policy Enforcer functionality include, but are not limited to:

- A standalone enabler implementation that uses other standalone enabler implementations to evaluate and enforce policies. Such an enabler implementation would be deployed as a separate component from other enabler implementations (see Figure 5, Case 3a and 3b).

NOTE: The "interceptor" (Figure 5, Case 2c, 3b) is a functional component that intercepts a request, generates the appropriate requests to a PEEM enabler implementation via the PEEM callable interface I0 and proceeds based on the result by letting the request reach its target, blocking the request or returning an error message. The "interceptor" function can be provided through a proprietary implementation, or through an implementation based on a future specification (The "interceptor" function has not been specified by OMA).

- In the deployment as depicted in Figure 5, Case 2b and 2c.Policy Enforcer functionality forms an integral part of the enabler implementation and is therefore not directly available to perform policy evaluation and enforcement for any other enabler implementations. In this case, the Policy Enforcer implementation performs its functionality and then passes execution control to the bundled enabler implementation. The Policy Enforcer implementation is not designed to pass execution control back to the implementation that invoked it, or forward to any implementation other than the one it is bundled with.

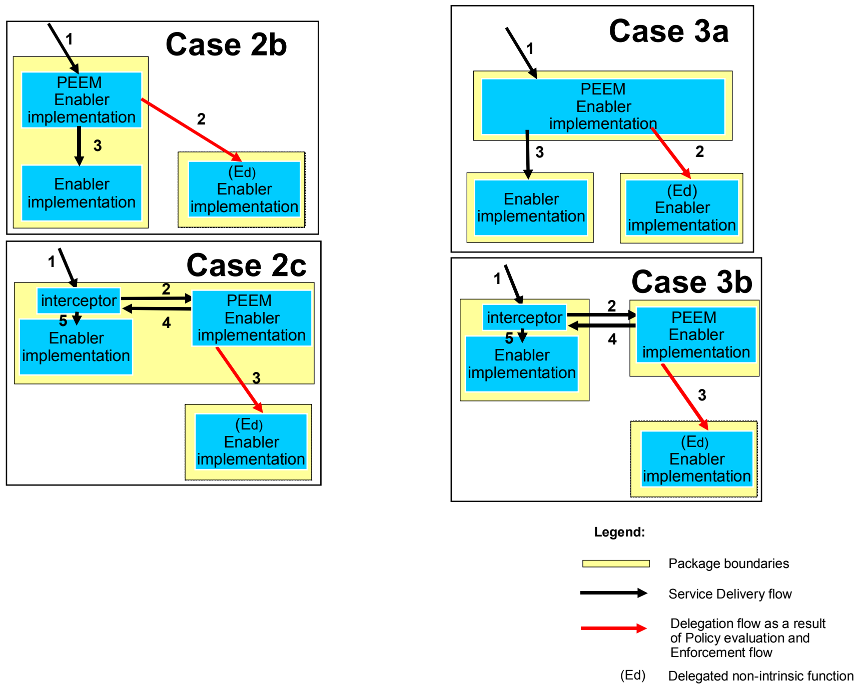
Case 2b
1
PEEM Enabler implementation
2
3
Enabler implementation
(Ed) Enabler implementation
Case 2c
1
interceptor
2
4
5
PEEM Enabler implementation
Enabler implementation
3
(Ed) Enabler implementation
Case 3a
1
PEEM Enabler implementation
3
2
Enabler implementation
(Ed) Enabler implementation
Case 3b
1
interceptor
2
4
5
PEEM Enabler implementation
Enabler implementation
3
(Ed) Enabler implementation
Legend:
Package boundaries
Service Delivery flow
Delegation flow as a result of Policy evaluation and Enforcement flow
(Ed) Delegated non-intrinsic function

Figure 5 - Target Policy Enforcer deployments (with flows)

6. Implications on enabler specification writers (normative)

The OMA Architecture Requirements [ARCH-REQ] state that enabler specifications should reuse existing specifications where possible. This approach includes reuse of existing OMA enabler specifications whenever possible (e.g. reuse of presence and group management enablers by the PoC enabler).

- **If applicable, an enabler MUST specify or reference one or more interfaces for its intrinsic functionality that will be used to interface to (i.e. invoke) its functions.**
- **If an enabler depends on already defined OMA functions, it MUST identify which other enablers' intrinsic functionality it will invoke to perform these already-defined OMA functions.**
- **An enabler MUST specify or reference only the functions, protocols and invocations that are essential (i.e. core) to its purpose**

Any requirements or features that are not intrinsic to an enabler should not be specified within the enabler's specification. An enabler's specification should only specify the intrinsic functionality required to fulfil its actual function.

For example, some enablers require having an identifier for the requesting entity. The requirement to perform the enabler's function is that there be a way to distinguish one requestor from another. It is not a requirement for the requestor's identity be verified using any particular mechanism (e.g. password, certificate, biometrics). The need to authenticate the requestor is a policy statement under the control of a domain. It is not required to perform the function of the enabler. Therefore, the authentication process is outside the scope of the enabler specification, allowing it either to be implemented as an added value by the enabler implementation or left to the policy enforcer implementation.

7. Security

An OSE domain may be protected using security mechanisms designed to reduce the risks associated with security threats. Which threats are important depend on application risk and security policy considerations.

The following security sections summarize the high level considerations that influence the use of the security mechanisms. This includes the threats and their associated risks that must be considered to create a security policy. The security policy defines which threats are likely and important, to what extent (the risks associated), which security requirements must be met, and what system considerations must be addressed. Such a policy should address system wide requirements for privacy and availability as well as deployment architecture considerations.

7.1 Security Threats

Security technologies are used to manage the risk and vulnerability associated with security threats (attacks taken on the systems, the information and data, and the services). The costs associated with the risks and costs of handling the vulnerabilities justify the cost of the security mechanisms. Security mechanisms are deployed to countermeasure the vulnerability by reducing the risks of the threat (e.g. risks of known attacks). The following list describes common security threats:

- Inappropriate **content modification** is a threat, either due to a malicious attack or due to an inadvertent mistake. Although a checksum can detect a change, it cannot detect tampering since the checksum may also be modified. Technologies such as digital signatures or Message Authentication Codes (MAC) (such as a keyed hash) may be used to detect changes and support source authentication. Such technologies may be deployed to protect information in transit (SSL/TLS), end-to-end at the application-messaging level (for instance, using WS-Security) or end-to-end at the application content level (for instance, using XML Digital Signature).
- **Denial of service** is an attack to either disable or degrade the ability of a server to provide services to clients. Overwhelming the server with requests that require excessive processing or that consume excessive resources, are two examples. Denial of service is the condition when a service falls below the required committed level, including unavailability of the services. Such denial of service may be cause by an intentional attack or by accidental conditions. Availability is a condition in which there is no denial of service or degraded communication quality.
- **Eavesdropping** is where information is viewed that should not be, either by examining messages in-transit or by examining content stored at a server. Using confidentiality features such as encryption of data or messages may prevent this. Encrypting data in transit, such as by using SSL/TLS, does not protect it when stored at a server or routed through application level intermediaries.
- A **man-in-the-middle** attack may be used to add, remove and change messages between two parties. Requiring authentication of both end parties may be used to avoid this problem.
- A **masquerade attack** hides the actual entity and impersonates to be a different entity that may have the authorization and privileges to access resources. This attack is usually used with reply and content modification. For example, authentication information can be captured and replayed after a valid authentication sequence has taken place.
- A **replay attack** is when someone captures and resends a message to obtain an anticipated result. Including some freshness material with messages, such as a timestamp or a unique non-repeating value, and checking this material before acting on a message at an endpoint can prevent this.
- **Trojan Horse** attacks have introduced quite significant impact in recent years. When introduced to the system, a Trojan horse performs an unauthorized function within its authorized function. One of the examples is the virus and worm attack.

7.2 Security Functions

The security functions described in this section describe the traditional security goals of reducing vulnerabilities of information, assets and resources. Important security functions include confidentiality, integrity, authentication, authorization, access control, non-repudiation, key management and security policy.

7.2.1 Authentication

Authentication is used to verify that a party is whom they assert to be and may be used, for example, to identify the sender of a message, a recipient, or the signer of some content.

Mutual authentication (the authentication of both parties in an exchange) is necessary to avoid man-in-the-middle attacks, and the use of timely information such as challenge response should be used to avoid replay attacks.

One widely accepted mechanism to authenticate communicating parties is the use of X.509 certificates with SSL/TLS for server authentication. SSL/TLS also allows the server to require client certificate-based authentication. This mechanism allows parties to authenticate to each other, assuming certificate management is handled properly. Credentials associated with authentication may be short or long-lived. If long-lived, then validation of credentials such as certificates is required of a recipient to ensure that revocation has not occurred. This may be done using OCSP, XKMS or CRLs to give some examples.

SSL/TLS may also further protect HTTP basic or digest authentication as well as application username and password authentication by providing integrity and confidentiality services.

7.2.2 Data Integrity

Integrity of information refers to the ability of a receiver to detect whether the content has been changed since creation, either maliciously or by accident. A checksum is not enough, since it could be maliciously replaced to mislead. Instead, a much stronger mechanism such as a digital signature or a MAC with the use of keying material can be used for the detection of any change in the content.

7.2.3 Confidentiality

Confidentiality is the property that unauthorized parties cannot view information. Typically confidentiality is provided using encryption technologies, such as symmetric and asymmetric encryption. The topic of confidentiality includes the choice and specifications of encryption algorithms, packaging of encryption metadata with encrypted content, and the relationship to the content and protocol model. Confidential communications are often necessary to preserve the privacy of information.

7.2.4 Key Management

The security and reliability of any communication process is directly dependent on the quality of key management and protection afforded to the keys. The functions of key management are to provide secure key generation, storage, renewal, revocation, exchange and use. The security of encrypted or authenticated data is strictly dependent upon the prevention of unauthorized disclosure, substitution, deletion and use of keys. If keys are compromised, the security of the data can no longer be assured.

Key management includes establishing a security context for creating, registering, sharing and validating keys. Key sharing can be performed differently depending on application requirements, including out of band communication. Scalable solutions may require a back end infrastructure, such as a public key infrastructure (PKI) or a Kerberos system. Differences in the methods and technologies result in different mechanisms, but the goals are the same, to reduce the risks of inappropriate key use and to provide a uniform, scalable system for key management.

7.2.5 Access Control/Authorization

Access Control and Authorization are security mechanisms that provide the appropriate access to a system or application. They may also be provided at different levels of the protocol stack. The network may make coarse-grained decisions about access to the network, systems may provide services to manage access to their resources, or the resources themselves may restrict who is able to use them. In some topologies, an authorization server may determine whether an authenticated party is allowed to access a resource or perform some action.

7.2.6 Non-Repudiation

Repudiation is defined as the "*Denial by one of the entities involved in a communication of having participated in all or part of the communication*" (Source: [X800]). Non-repudiation is the use of technology, business rules and legal mechanisms to reduce the risk of repudiation to an acceptable level.

Discussion of non-repudiation in a pure technology sense is not meaningful since the issue is intrinsically linked to business and legal issues. Non-repudiation technologies can be correctly considered to support dispute resolution and support for reduction of repudiation risk.

Endorsement using long-lived digital signatures may be used to provide evidence that the signing party has agreed to a contract, approved an action, read some material or agreed to some other statement (verbal or written) when creating the signature. Non-repudiation requires that only the signer have access to their signing material, that appropriate information is included with the signature (such as a timestamp and the reason for signing) and that the signature be persistent. This means that signatures for non-repudiation cannot be transitory signatures such as used in SSL/TLS, but must be long-lived signatures suitable for dispute resolution.

8. Migration from OMA *silo* enabler architectures towards the OSE using Policy Enforcement

8.1 Relationship between exposed interfaces and policies at deployment

When deployed, regardless of its realization, the Policy Enforcer has to support evaluation and enforcement of the domain owner’s policies in a variety of scenarios.

Deployed entities in the OSE may interact with each other differently, depending on how the controlled exposure of an enabler implementation is achieved (e.g. the existence of policies to be applied on requests, the content of the policies, the existence of additional parameters (P) required to satisfy policies). **Figure 6** illustrates how interfaces exposed depend on the content of the domain policies for particular enabler implementations.

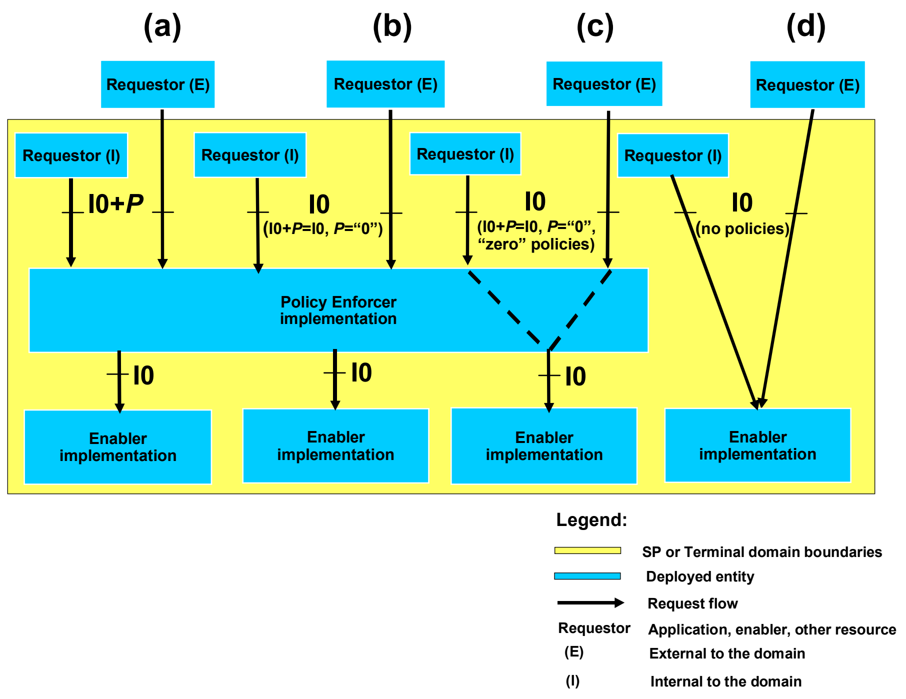
(a)
(b)
(c)
(d)
Requestor (E)
Requestor (I)
I0+P
I0
(I0+P=I0, P=“0”)
(I0+P=I0, P=“0”, “zero” policies)
(no policies)
Policy Enforcer implementation
Enabler implementation
Legend:
SP or Terminal domain boundaries
Deployed entity
Request flow
Requestor Application, enabler, other resource
(E) External to the domain
(I) Internal to the domain

Figure 6– Interfaces exposed in relationship to policies

Any combination of the represented scenarios (a) through d)) may occur separately, or simultaneously in a deployment:

a) Illustrates the case where a Policy Enforcer implementation protects an enabler implementation from requests/responses, when P parameters are required to satisfy existing domain policies.
b) Illustrates the case where a Policy Enforcer implementation protects an enabler implementation from requests/responses, when P parameters are *not* required to satisfy existing domain policies. In this case, the I0+P exposed to the requestors by design is actually the same as the I0 of the target enabler implementation (P=”0”).
c) Illustrates the case where a Policy Enforcer implementation is intended to protect an enabler implementation from requests/responses, but only “zero policies” are present for that enabler implementation (see explanation for “zero policy” in section 5.4.2). In this case, the I0+P exposed to the requestors by design is actually the I0 of the target enabler implementation (P=”0”). The dashed lines represent the fact that the Policy Enforcer does not process such request/response.
d) Illustrates the case where a Policy Enforcer implementation is not needed (no policies exist for some enabler implementations).

8.2 Enabler implementations and deployments

NOTE to the Reader: This section contains information about OMAs proposed Policy Evaluation, Enforcement and Management (PEEM) enabler. The information in these sections describes work in progress.

An enabler implementation can invoke any standardized function such as authentication, charging or Group Management, which are required to satisfy the enabler specification (i.e. the principle of delegation and reuse). Some of these function invocations may be triggered as a result of a policy decision. The enabler implementation can accomplish those policy triggered function invocations (e.g. authorization) either by:

- Implementing the function (e.g. authentication) itself (Figure 7, Case 1);
- Performing the policy evaluation and enforcement itself by invoking a separate (modular) implementation that performs the policy processing function. Figure 7, Case 2a makes use of a constrained policy evaluation and enforcement mechanism where the vendor supplying the enabler implementation determines which operations (i.e. policies) the enabler implementation can invoke (i.e. there is a built-in, non-changeable selection of policies to be evaluated and enforced). Figure 7, Case 2b illustrates a full policy evaluation and enforcement mechanism that allows the domain to determine which operations (i.e. policies) the enabler implementation invokes. In this case the policy evaluation and enforcement mechanism is applied in proxy mode. Figure 7, case 2c illustrates a variant to case 2b in the sense that it illustrates that the policy processing mechanism is applied in callable mode;
- Delegating the invocation to a policy processing entity that will invoke a separate (modular) implementation that performs the required operation Figure 7, Case 3, where case 3a illustrates the case where the policy evaluation and enforcement mechanism is applied in proxy mode and 3b illustrates the case where the policy processing mechanism is applied in callable mode.

To summarize the distinctions between these choices:

- For Figure 7, case 1, the implementation of the operations is done in the enabler implementation;
- For Figure 7, case 2a, 2b and 2c, the implementation invokes other separate components to perform the operations, which allow all enabler implementations in the deployment to use the same operation and enabler implementations and reduce the silo effect;
- For Figure 7, case 3a and 3b, the implementation invokes a separate component to perform the policy processing, which itself may invoke separate components to perform the operations.

Figure 7, Cases 1 and 2a are consistent with the OSE Policy Enforcer described earlier and correspond to the current *silo* situation.

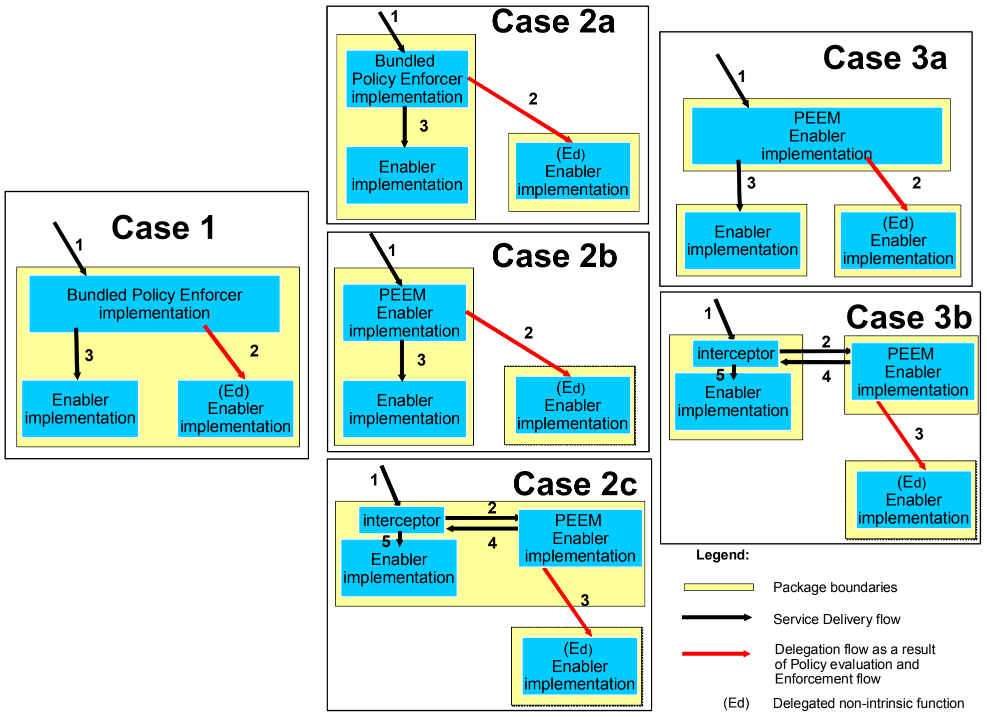
Case 1
1
Bundled Policy Enforcer implementation
3
2
Enabler implementation
(Ed) Enabler implementation
Case 2a
1
Bundled Policy Enforcer implementation
2
3
Enabler implementation
(Ed) Enabler implementation
Case 2b
1
PEEM Enabler implementation
2
3
Enabler implementation
(Ed) Enabler implementation
Case 2c
1
interceptor
2
4
5
PEEM Enabler implementation
Enabler implementation
3
(Ed) Enabler implementation
Case 3a
1
PEEM Enabler implementation
3
2
Enabler implementation
(Ed) Enabler implementation
Case 3b
1
interceptor
2
4
5
PEEM Enabler implementation
Enabler implementation
3
(Ed) Enabler implementation
Legend:
Package boundaries
Service Delivery flow
Delegation flow as a result of Policy evaluation and Enforcement flow
(Ed) Delegated non-intrinsic function

Figure 7 - Examples of Policy Enforcer deployments (with flows)

8.3 Migration through the use of PEEM

In addition to the Policy Enforcer deployments using OMA enablers as described in Figure 5, Cases 3 and 2b in Section 5.4.4, the current OMA *silo* architecture permits vendors to implement their own policy evaluation and enforcement according to the following deployment options:

- A bundled Policy Enforcer implementation that uses other separate enabler implementations to evaluate and enforce policies. The bundled Policy Enforcer enabler implementation is an integral part of another enabler implementation and is not available to perform policy evaluation and enforcement for any other enabler implementations. The Policy Enforcer implementation does not have the full generality of an PEEM enabler implementation, e.g. the choice of policies to evaluate and enforce might be determined at implementation design time (see Figure 7Case 2a);
- A bundled Policy Enforcer implementation that performs its own policy evaluation and enforcement without using other enabler implementations to evaluate or enforce policies. The Policy Enforcer implementation does not have the full generality of a PEEM enabler implementation, e.g. the choice of policies to evaluate and enforce might be determined at implementation design time (see Figure 7, Case 1).

All four identified cases, as described in Figure 7, map into the OSE logical architecture and associated flows. Cases 1 and 2a map into the OMA *silo* enabler architectures, where Cases 2b and 3 represent the OSE architecture and enablers.

As a result, PEEM is logically present across each reference point, as described in [ARCH-INVEN]. Specific Service Provider deployments may not require any policies to be processed, in which case the domain may not deploy a PEEM enabler implementation.

In some cases, specific domain deployments may require policies to be processed by PEEM only across some Reference Points, in which case the domain may choose to deploy a PEEM enabler implementation only on those Reference Points.

Without requiring any changes to existing enabler specifications, domains can introduce an implementation of the PEEM to perform policy processing operations that do not conflict with existing enabler implementations. For example, an enabler may specify its own methodology for ensuring security, which means that conforming enabler implementations must implement the defined security methodology. However, PEEM could be used for functions not defined by the enabler and not provided by the enabler implementation.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-Service_Environemt-V1_0	07 Sep 2004	Initial document to address the basic starting point Ref TP Doc# OMA-TP-2004-0299-OSE-Approval
OMA-Service-Environment-V1_0_1	14 Jun 2005	Incorporates inputs: ▪ OMA-ARC-2005-0150R02-LATE0OSE-V1.0-clarifications-for-Policy-Enforcer ▪ OMA-ARC-2005-0152-OSE-v1-changes-from-0090-and-0118 ▪ OMA-ARC-2005-0154R01-Conceptual Model and architecture (to clarify in which sections the OSE architecture is described) ▪ OMA-ARC-2005-0164-revise OSE v1&2 section 5.4.4 (text deletion) ▪ OMA-ARC-2005-0165-CR-Apply-Doc147-to-OSE-v1 (adds text to section 5.2.2 (Enabler Implementation) that clarifies an enabler may be defined) ▪ OMA-ARC-2005-0173-CR-OSEv101-Definitions (correction of abbreviations PE and PEEM in section 3.3) ▪ OMA-ARC-2005-0181-Update-to-changes-for-I0-and-I0+P (explanations update to definition for Parameter Pand explanations for I0 and I0+P) ▪ OMA-ARC-2005-0186-OSE-Deployment-options (implementation options that make use of the PEEM enabler in a *proxy* mode) ▪ OMA-ARC-2005-0187-Editorial-changes-over-181 (editorial changes to the explanations of I0 and I0+P) ▪ OMA-ARC-2005-0190-OSE-deployment-relationship-of interfaces-to-policies (new text and figure 6 to OSE V1.0, via a new section 7.1) ▪ OMA-ARC-2005-0166R03-Clarify-composed-interfaces (section of 5.4.1 of the OSE AD enabler composition by the PE) ▪ OMA-ARC-2005-0189R01-CR-Align-Figure1-Caption.doc (changes to align two Change Requests: docs 152 and 154R01) Ref TP Doc#OMA-TP-2005-0183-Notification-of-changes-OSE-V1_0_1.doc
OMA-Service-Environment-V1_0_2	26 Jul 2005	Minor update (some typos + additional wording to avoid misunderstandings) OMA-ARC-2005-0220R01-CR-OSEv101-section5 OMA-ARC-2005-0265-typo-cleanup-OSE
	03 Aug 2005	Minor update: fixed cross-references in Sections 4, 5 and 7.
OMA-AD-Service-Environment-V1_0_3	22 Jun 2006	Changed to AD document type + minor updates: OMA-ARC-2006-0070-Removal-of-reference OMA-ARC-2006-0206-OSEv1-apply-new-defs
OMA-AD-Service-Environment-V1_0_4	06 Dec 2006	Incorporates input: - OMA-ARC-2006-0378R01-CR_OSE_copy_AD_V2_changes_to_V1
	01 Feb 2007	2007 Copyright Minor update: OMA-ARC-2007-0009-CR_OSE_AD_changes_to_address_CONR

Appendix B. Deriving an OMA Service Environment architecture

The OSE architecture contains a set of interfaces that are specified by OMA. The Architecture requirements document [ARCH-REQ] implies the need for a set of interfaces. These interfaces could be implemented in various ways, e.g. as one component (software module) for each interface, one single component implementing all interfaces, or a mixture of these two options.

The following interfaces are derived from the Architecture requirements document [ARCH-REQ]. Each interface is cross-referenced to one or several Architecture requirements.

Derived OMA Architecture Interface:

1. Interface for operations and management (Cross referenced with [ARCH-REQ] 6.3.3#1)
2. Interface for the discovery of service enablers (Cross referenced with [ARCH-REQ] 6.3.2#1; 6.3.2.1#3, #5; 6.1.3#11)
3. Interface for the registration of service enablers (Cross referenced with [ARCH-REQ] 6.3.2.1#4, #5)
4. Interface for the discovery of services (Cross referenced with [ARCH-REQ] 6.3.2.1#2)
5. Interface for the registration of services (Cross referenced with [ARCH-REQ] 6.3.2.1#1)
6. Interface for discovery of conditions for the use of service enablers (Cross referenced with [ARCH-REQ] 6.1.3#11)
7. Interface towards a policy management mechanism (Cross referenced with [ARCH-REQ] 6.1.3#12; 6.1.5#5)
8. Interface to provision services, service enablers and user parameters (Cross referenced with [ARCH-REQ] 6.1.5#4)
9. Interface for subscription management (Cross referenced with [ARCH-REQ] 6.1.3#13)
10. Identity management mechanism associating device identification with federated identity (Cross referenced with [ARCH-REQ] 6.1.3#8, 9, 10; 6.1.1#11)
11. Interface to network exposing network characteristics (Cross referenced with [ARCH-REQ] 6.1.3#8)
12. Interface to charging (to gather accounting and charging information) (Cross referenced with [ARCH-REQ] 6.1.2#2)
13. Interface to authentication function (Cross referenced with [ARCH-REQ] 6.1.1#1)
14. Interface to authorization function (Cross referenced with [ARCH-REQ] 6.1.1#14)
15. Interface from authorization function to charging enabler (and the reverse) (Cross referenced with [ARCH-REQ] 6.1.1#14)
16. A method to connect between identity, authorization, and authentication components, e.g. cookies or other session tokens (Cross referenced with [ARCH-REQ] 6.1.1#14)

Policy (constraints) in all interfaces (Cross referenced with [ARCH-REQ] 6.1#16)

Access to "back-end systems" (charging, accounting, payment, provisioning, Operations & Management, etc.) can be realised by interfacing these through a component, and using the standard OMA interfaces between the enabler and the component.

Analysed OMA Architecture requirements:

- ([ARCH-REQ] 6.1# 16) When authorized, Principals MUST be able to set policies (e.g. charging policies and privacy policies) on any request (including discovery)
- ([ARCH-REQ] 6.1.1#1) The OMA Service Environment MUST provide mechanisms for authentication of users, applications and third-party Service Providers, and authorization for the use of service enablers across and within Service Provider domains.

- ([ARCH-REQ] 6.1.1#5) The OMA Service Environment MUST enable single sign-on and single log-out to span enablers in a single domain or across multiple Service Provider domains. One-time authentication or a SSO MUST remain valid throughout a continuous session
- ([ARCH-REQ] 6.1.1#11) The OMA Service Environment MUST support a mechanism to federate and de-federate identity information across Service Provider domains.
- ([ARCH-REQ] 6.1.1#14) The OMA Service Environment MUST provide an interface between the authorization function and the charging enabler.
- ([ARCH-REQ] 6.1.2#2) The OMA Service Environment MUST provide an interface where Accounting and Charging information is to be gathered.
- ([ARCH-REQ] 6.1.3#3) The OMA Service Environment MUST enable the communication of service monitoring data (e.g. performance measurements) between actors.
- ([ARCH-REQ] 6.1.3#5) The OMA Service Environment MUST provide the means to manage the activation, registration, authentication, and authorization of users and service components.
- ([ARCH-REQ] 6.1.3#8) The OMA Service Environment MUST provide a mechanism by which device and network information can be communicated to an authorized third-party (with respect to the information holder) in a manageable way. This mechanism MUST allow for the automated discovery of new devices and new characteristics in existing devices.
- ([ARCH-REQ] 6.1.3#9) The OMA Service Environment MUST provide a mechanism to enable Third-Parties to obtain an identification for an end-user who uses a particular device to access authorized third-party applications.
- ([ARCH-REQ] 6.1.3#10) The OMA Service Environment MUST provide a mechanism to allow Third-Parties to discover the device(s) currently used by an end-user, if registered on a network (e.g. where to send a notification to the employee).
- ([ARCH-REQ] 6.1.3#11) The OMA Service Environment MUST provide a mechanism for an authorized third-party to discover the conditions for using a service enabler exposed by a particular Service Provider in a dynamic manner.
- ([ARCH-REQ] 6.1.3#12) The OMA Service Environment MUST support a mechanism for Service Providers and other authorized actors to enforce the conditions for use of a service enabler.
- ([ARCH-REQ] 6.1.3#13) The OMA Service Environment MUST have a single logical point that handles subscriber and subscription information.
- ([ARCH-REQ] 6.1.5#4) The OMA Service Environment MUST provide a common mechanism for Provisioning of services, service enablers and user parameters.
- ([ARCH-REQ] 6.1.5#5) The OMA Service Environment SHOULD provide a mechanism to manage and use policies (e.g. access policies, charging polices, service level agreements, etc.).
- ([ARCH-REQ] 6.3.2#1) The OMA Service Environment MUST have a single logical access point (e.g. Common Directory) to handle: 1) registration, 2) discovery and 3) functions and data that handle information relevant to more than one single service enabler.
- ([ARCH-REQ] 6.3.2.1#1) The OMA Service Environment MUST support Service Registration for Services visible to the end-user.
- ([ARCH-REQ] 6.3.2.1 #2) The OMA Service Environment MUST support Service Discovery for services visible to the end user.
- ([ARCH-REQ] 6.3.2.1#3) The OMA Service Environment MUST support Discovery for an implementation of a Service Enabler.

- ([ARCH-REQ] 6.3.2.1#4) The OMA Service Environment MUST support Registration for any implementations of a Service Enabler.
- ([ARCH-REQ] 6.3.2.1#5) Within the OMA Service Environment it MUST be possible to register, discover, and retrieve information (e.g. a service enabler's address) using a resource identifier (e.g. a user identifier).
- ([ARCH-REQ] 6.3.3#1) The OMA Service Environment MUST define a common interface for the operations and management (O&M) of both common and service-specific enablers or applications (including service monitoring and end-to-end service delivery).

Appendix C. Reference Points versus Interfaces

It is possible to model interactions between architectural entities by means of:

Interfaces: Interfaces solely focus on how the resource can be interacted with, independently of what interacts with the resource (see Figure 8).

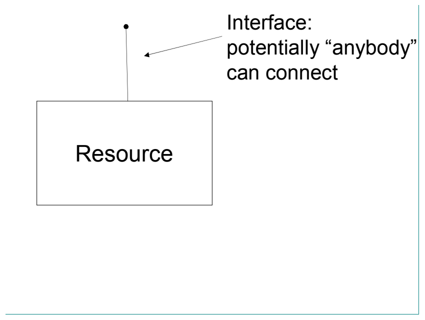

Figure 8 - Schematic view of an interface

Reference Points: Reference points explicitly enumerate the end points that can interact with the resource. A Reference Point is a conceptual point at the conjunction of two non-overlapping functional groups (source: [ITU-T I.112]). It consists of none or any number of interfaces of any kind. This means a Reference Point can host more than one transport protocol or payload. If a Reference Point is defined between two architectural entities, it does not necessarily require an interface (transport protocol, payload, API, etc.) to be associated at all. This means the two architectural entities can communicate using any protocol over any interface (it is not defined, but the communication relationship exists).

There is always only one or no Reference Points between the same two architectural entities, no matter how many interfaces, protocols or APIs may exist between the two (see Figure 9).

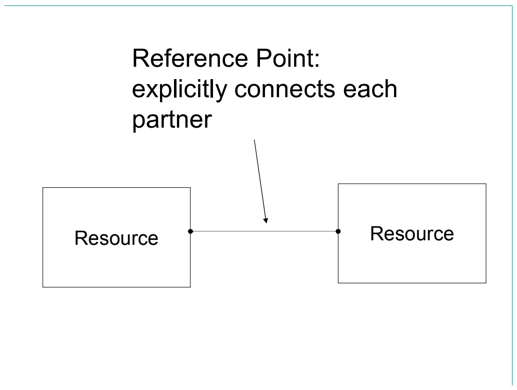

Figure 9 - Reference Point schematically

Reference points are commonly used in communities that specify, design, implement or deploy network-level systems (e.g. Telecommunications environments). Communities that specify, design, implement, or deploy software systems rather rely on interface descriptions (e.g. IT environments).

However, the two approaches provide equivalent views of the system either through the interfaces that it exposes or through reference points that typically explode each interface into multiple reference points; one per end point / architectural entity that can interact with the system through that interface. References points (between two end points) that support multiple transport protocols map to one interface with multiple interface realizations.

The relationship of interfaces to reference points is illustrated in Figure 10.

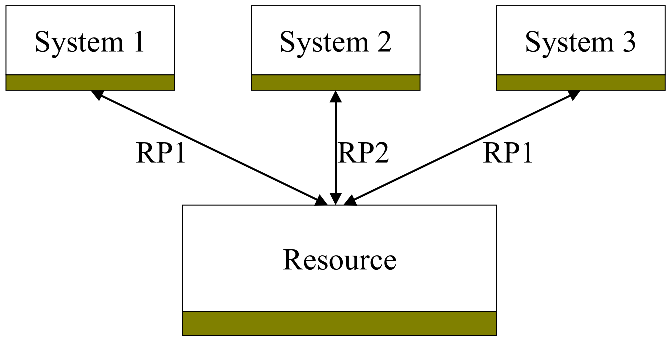

Figure 10 - Equivalency of interface point of view and reference point of view