



Contract: 027617

SPICE

Service Platform for Innovative Communication Environment

FP6 - Integrated Project (IP)

Priority T2 – Information Society Technologies

WP 2

Deliverable 2.5

Exposure Mechanisms to Access Generic Service Enabler
Components

Due date of deliverable: [30/06/2007](#)

Actual submission date: Updated [18/07/2008](#)

Start date of project: 01/01/2006

Duration: 30 Months

Project coordinator name: Christophe Cordier

Organization name of lead contractor for this deliverable: [Ericsson](#)

Revision: [d1.04](#)

PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE **SPICE** CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE **SPICE** CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT

Glossary

Action	An operation on a resource [1]
Discovery Facility	System entity providing service directory with semantic search capability
Environment	The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource or action [1]
Policy	A set of rules, an identifier for the rule-combining algorithm and (optionally) a set of obligations [1]. Without going into details (rule, target, effect, condition), policies declare who (subject) may do what (action) on what (resource) under what circumstances (conditions, environment).
Policy Administration Point	The system entity that creates a policy or policy set [1]
Policy Decision Point	The system entity that evaluates applicable policy and renders an authorization decision [1]
Policy Enforcement Point	The system entity that performs access control, by making decision requests and enforcing authorization decisions [1]
Policy Information Point	The system entity that acts as a source of attribute values [1]
Resource	Data, service or system component [1].
Service component	Elementary building block providing either SPICE end user service logic or operational management logic. Service components expose web services interfaces by which they are accessible and conform to conventions that make them manageable by the platform as components (e.g. lifecycle operations).
Security Gateway	An element residing on the border of the platform and implementing the necessary security services (authentication of origin, data integrity, confidentiality, anti-replay protection). All traffic between a SPICE platform and another party flows through it.
Subject	An actor whose attributes may be referenced by a predicate [1]

List of authors

Full Name	Company Information
Jan van der Meer	Ericsson
Mathieu Hutschemaekers	Ericsson
Gábor Paller	Nokia Siemens Networks
Gerhard Fisher	SAGO
Gábor Márton	Nokia Siemens Networks
Davide Cipolla	IRIS

Document Information

Document Name:	Exposure Mechanisms to Access Generic Service Enabler Components
Document ID:	SPICE_D2_5_Ericsson
Revision:	D1.04
Revision Date:	18/07/2008
Author:	Mathieu Hutschemaekers
Security:	

Approvals

	Name	Company	Date	Visa
Technical Coordinator	Christophe Cordier	FT	18/07/2008	
WP leader	Gert-Jan Rijckenberg	Ericsson	30/06/08	
Technical Manager	Mathieu Boussard	ACIT	18/07/2008	
Quality Manager				

Documents history

Revision	Date	Modification	Authors
d.01	06/06/2007	First version	M. Hutschemaekers
d.04	19/06/2007	Extended introduction	M. Hutschemaekers
d0.12	25/06/2007	Exposure introduced	G. Paller
d0.13	26/06/2007	Access control and IM introduced	G. Márton, D. Cipolla
d0.14	26/06/2007	Review chapter 5	J. van der Meer
d0.15	27/06/07	Added Conclusion	M. Hutschemaekers

d0.16	28/06/07	Rework from internal review	M. Hutschemaekers
d0.17	09/07/07	Rework from review technical manager	M. Hutschemaekers
d0.18	12/07/07	General rework, rework from review M. Belaunde	M. Verheijen, M. Hutschemaekers
D1.01		Uodate security section	G. Márton
D1.02	19/05/08	Update Call control enabler, added the presence enabler, updated the third-party components section,	M. Hutschemaekers
D1.03	26/06/08	Replace Discovery Facility with Service Broker interface; introduce front end component for statistics; editorials	M. Hutschemaekers
D1.04	30/06/08	Editorial comments	M. Hutschemaekers



TABLE OF CONTENTS

1	<u>INTRODUCTION</u>	10
1.1	PURPOSE OF THIS DOCUMENT	10
1.2	STRUCTURE OF THE DOCUMENT	11
2	<u>GENERIC SERVICE ENABLERS</u>	13
2.1	INTRODUCTION	13
2.2	INSTANT MESSAGING	13
2.3	CALL CONTROL ENABLER	14
2.4	LOCATION ENABLER	15
2.5	PRESENCE ENABLER	17
3	<u>EXPOSURE MECHANISMS</u>	19
3.1	INTRODUCTION	19
3.2	EXPOSURE MECHANISM	19
3.2.1	ADAPTING SERVICE INTERFACES AND META-DATA	21
3.2.2	INTELLIGENT ADAPTOR INSERTION	23
3.3	CONCLUSION	25
4	<u>IMPACT ON THE SPICE PLATFORM</u>	26
4.1	INTRODUCTION	26
4.2	ACCESS CONTROL	26
4.2.1	MAIN CONCEPTS	27
4.2.2	TRUST MODEL	27
4.2.3	SECURITY SERVICES	28
4.2.4	IDENTIFICATION	29
4.2.5	POLICIES	29
4.3	ADDITIONAL STATISTICS	29
5	<u>SPICE IMPACT ON THE COMPONENT</u>	31
5.1	INTRODUCTION	31
5.2	REQUIREMENTS FOR THIRD-PARTY ACCESS	31
5.2.1	TRUST MODEL	31
5.2.2	SECURITY SERVICES	31
5.2.3	IDENTIFICATION	32
5.2.4	POLICIES	32
5.2.5	CONCLUSION	32
5.3	INTERFACE TO THE SERVICE BROKER	32

6 CONCLUSION AND OUTLOOK..... 34

7 REFERENCES 35

List of figures

Figure 1: Accessing the Service Broker to find a service.....	10
Figure 2: Using functional interfaces	11
Figure 3: Broker-based component binding.....	20
Figure 4: Broker-less service adaptation	20
Figure 5: Simplified architecture of the service meta-info adaptation system	21
Figure 6: The adaptor insertion problem.....	23
Figure 7: Interaction of adaptor insertion subsystem and the Discovery Facility	25
Figure 8: Main concepts of access control.....	27
Figure 9: X-SPICE-Asserted-Identities HTTP request header syntax.....	29
Figure 10: SPICE enabler with front end component.....	30
Figure 11: Security Gateways (SEGs), trusted third parties and public access	31

Abbreviations

Abbreviation	Signification
AAS	Authentication and Authorization Support
EE	Execution Environment
IdP	Identity Provider
IM	Instant Messaging
IMS	IP Multimedia Subsystem
KMF	Knowledge Management Framework
KPI	Key Performance Indicators
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
QoS	Quality of Service
RDF	Resource Description Framework, http://www.w3.org/RDF/
SC	Service Component
SCN	Service Component Network
SE	SPICE Element
SEG	Security Gateway
SLA	Service Level Agreement
UML	Unified Modeling Language
UPM	User Privacy Management
URI	Uniform Resource Identifier
XACML	Access Control Markup Language
XML	Extensible Markup Language
XSL	Extensible Style sheet Language
XSLT	XSL Transformations

1 Introduction

1.1 Purpose of this document

This document describes what is necessary for third-party services to access SPICE components. The impact of allowing this third-party access on both the SPICE platform side and the third-party service side is described. To illustrate the functions of SPICE enablers the interfaces of a number of SPICE enablers are provided.

A third-party service, i.e. a service outside SPICE that wants to use one of the SPICE enablers, needs to follow at least the following two steps:

1. Establish that the available enablers within SPICE can actually provide the wanted service.
2. Access the functional interfaces of the enablers selected as providers of the wanted service.

For the first step, matching wanted services with available services, SPICE offers the Service Broker. The Service Broker accesses the Discovery Facility, which allows discovery of service component meta-information, including the service component access mechanisms.

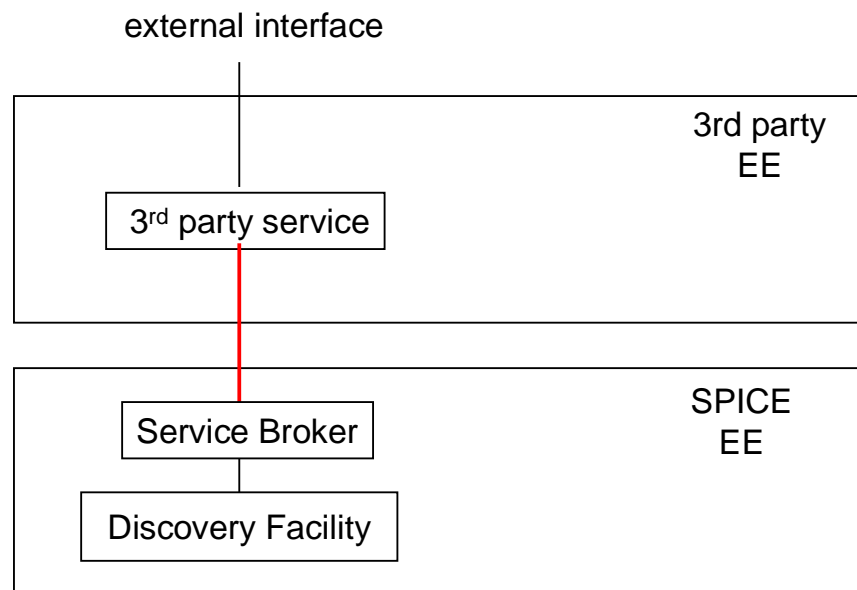


Figure 1: Accessing the Service Broker to find a service

Figure 1 shows that a third-party service in the third-party Execution Environment (EE) accesses the Service Broker to find a service fitting to the request. The challenge for the SPICE platform is to handle as many different service requests as possible. In other words, the challenge is to reject as few service requests as possible when the basic functionality is offered by one of the components in the SPICE platform. This is established through dynamic service adaptation. A new approach to service adaptation is included in this document.

For the second step, actually using the functional interfaces, it is important that no SPICE platform rules, especially not the ones regarding security, are violated. The platform rules

are described in the Cookbook [2]. They cover the implementation of security between components. This document extends the Cookbook Authentication and Authorization Mechanism section, and studies the implications when one of the components is an external component.

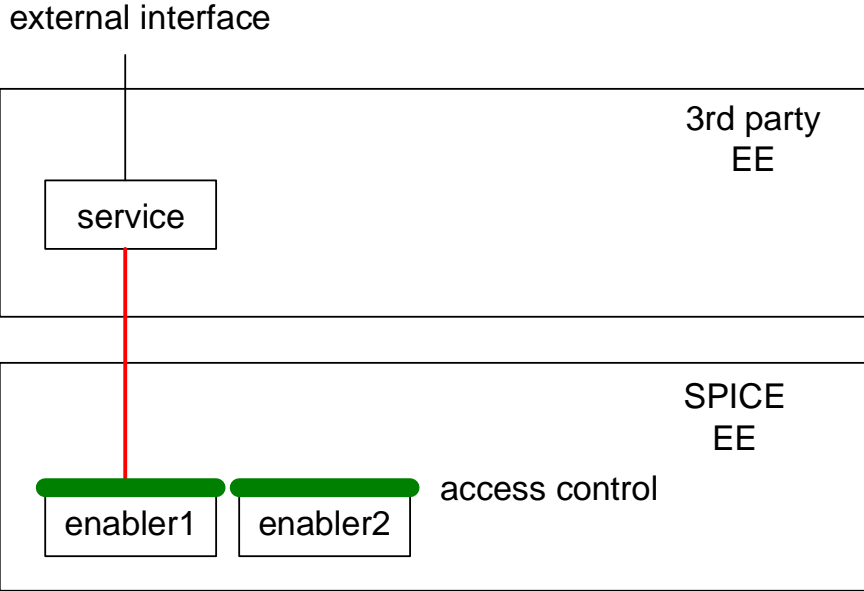


Figure 2 shows the actual usage of the functional interfaces of the respective services.

Figure 2: Using functional interfaces

The first approach to the subjects above is looked at from the SPICE platform point of view: which functions may be or need to be introduced to support third-party access. The second approach is seen from the third-party component point of view: which interfaces toward SPICE platform functions should be implemented by third-party services.

1.2 Structure of the document

Chapter 2 (Generic Service Enablers) describes the interfaces of the following components:

- Instant messaging server
- Call control server
- Location server
- Presence enabler

Chapter 3 (Exposure mechanisms) presents a different approach to dynamic service adaptation.

Chapter 4 (Impact on the SPICE platform) describes the mandatory and optional extensions to the SPICE platform for use with external components.

Chapter 5 (SPICE impact on the component) discusses which interfaces may or need to be used by the external services.

2 Generic Service Enablers

2.1 Introduction

In the descriptions below the following syntax is used for the signature:

```
operationName(ParameterType) : ResultType
```

or

```
operationName(ParameterType parametername) : ResultType
```

2.2 Instant Messaging

The basic Instant Messaging component is a SPICE component that provides the SIP Instant Messaging functionality. More specifically, this component can work with SIP Instant Messages (IMs) in two ways:

- As a sender, it can be used to send IMs to users.
- As a receiver, it can be programmed to listen for Instant Messages sent to a SIP User Agent associated with the component, and thus with the SPICE service containing the component. Whenever this User Agent receives an IM, the component will notify the SPICE service logic. The set of IMs that actually produce a notification can be narrowed to a subset that matches a filter, specified at subscription time. The messages not matching the filter will be silently discarded.

These features are exposed through a service-specific web service. The notification feature requires that the SPICE service logic provide a “callback” endpoint, in the form, for instance, of a web service.

The component and the “callback” interfaces are described in WSDL files. For the component part, all common features of a SPICE component are defined, together with the specific functionalities offered by the Instant Messaging Enabler.

The operations of the Instant Messaging component are defined as follows:

- **sendMessage**: this operation is used to send a SIP IM. The signature of the method is the following:

```
sendMessage(SendMessageRequest) : SendMessageResponse
```

The `SendMessageRequest` object contains strings for the source SIP URI, the destination SIP URI, the body of the message and a listener URL (reserved - currently unused). Moreover, it contains a Boolean value indicating whether a SIP ACK is expected from the destination SIP endpoint upon receipt of the message. The `SendMessageResponse` object contains a string describing whether the operation was successfully terminated.

- **subscribeToMessage:** this operation is used to subscribe to notifications for a particular set of messages received by the SIP User Agent associated with the basic component. The signature of the method is the following:

```
subscribeToMessage(SubscribeToMessageRequest) :  
SubscribeToMessageResponse
```

The `SubscribeToMessageRequest` object contains a `Filter` object and a string representing a listener URL. The `Filter` object defines a subset of the instant messages received by the SIP User Agent associated with the component, which will cause the component to send a notification to the SPICE service logic. The listener URL is, for instance, a web service URL, which is used to provide the “callback address”, at which the service logic expects to be notified by the IM component.

The `Filter` can define a subset of messages based on the following parameters:

- 1) The source SIP URI
- 2) The destination SIP URI
- 3) A regular expression for the body of the IM

The `SubscribeToMessageResponse` object contains a string representing the subscription ID that uniquely identifies the submitted subscription. This ID will be used in the notification.

The SPICE service logic callback operation is exposed as follows:

- **onMessage:** this operation can be called by the IM component to notify the SPICE service logic that a message matching the filter specified at subscription time has been received by the SIP User Agent associated with the IM component. The signature of the method is the following:

```
onMessage(OnMessageRequest) : OnMessageResponse
```

The `OnMessageRequest` object contains a `MessageEvent` object, which in turn contains strings for the source SIP URI, the destination SIP URI, the message body and the subscription ID.

The `OnMessageResponse` object contains a string describing whether the notification was accepted or rejected.

2.3 Call Control Enabler

The Call Control Enabler is a SPICE component that establishes calls as part of a service. The Call Control Enabler provides methods for

- Defining a call to be set up with called and calling parties
- Order the setup of a previously defined call

The operations of the Call Control component are defined as follows:

- **createReference:** This method provides a reference to the call to be set up between the two parties mentioned in the invocation. The method-signature is:

```
createReference(String, String) : String
```

The resulting `String` allows the invocation of the method below.

- **activateCall:** This method arranges the setup of the call previously defined. The signature of the method is:

```
activateCall(String) : String
```

2.4 Location Enabler

The Location Enabler is a SPICE component that can provide context information related to the geographical location of a user. This feature is exposed through a service-specific web service interface, like every component in the SPICE platform. The component interface is described in a WSDL file, in which all common features of a SPICE component are defined, together with the specific functionalities offered by the Location Enabler.

The features of the Location Enabler are the following:

- **getPosition:** this method provides the position (latitude, longitude and accuracy of location) related to a specific GSM or UMTS mobile cell. The signature of the method is:

```
getPosition(SpiceLocationRequest) : SpiceLocationResponse
```

The `SpiceLocationRequest` object contains the cell identifier and (if applicable) the power measurement detected by the mobile device. The `SpiceLocationResponse` object contains the position, namely latitude, longitude and accuracy. This signature specification is based on an implementation by Telecom Italia.

- **getCompleteLocation:** this method provides the position - latitude, longitude and accuracy of location - and the civil address related to a specific GSM or UMTS mobile cell. The method signature is:

```
getCompleteLocation(SpiceLocationRequest) : SpiceLocationResponse
```

The `SpiceLocationRequest` object is the same structure defined in the previous method. The `SpiceLocationResponse` object contains the position (latitude, longitude and accuracy) and, if available, the civil address (country, city, street, building, floor, room, ...). Like the previous method, this signature specification is based on an implementation by Telecom Italia..

- **getCivilAddress:** this method provides the civil address related to a set of geographic coordinates (geo-coding). The signature of the method is:

```
getCivilAddress(SpiceLocationInfoRequest) : SpiceLocationResponse
```

The `SpiceLocationInfoRequest` object contains latitude and longitude expressed in WGS84 format. The `SpiceLocationResponse` object contains the civil address that can be associated with that position.

- **getGpsPosition:** this method provides the position (latitude and longitude) related to a civil address (reverse geocoding). The method signature is:

```
getGpsPosition(SpiceLocationInfoRequest) : SpiceLocationResponse
```

The `SpiceLocationInfoRequest` object contains the civil address information. The `SpiceLocationResponse` object contains the position related to `SpiceLocationInfoRequest`.

- **getGpsRoute:** this method provides a route between two geographical points. The signature of the method is:

```
getGpsRoute(SpiceRouteRequest) : SpiceRoute
```

The `SpiceRouteRequest` object contains departure and arrival points. The `SpiceRoute` object contains the distance between those two points, the total traveling time (including possible distances on foot), the driving time and a list of steps describing the route.

- **getGpsPoi:** this method provides a set of Points Of Interest per category. The method signature is:

```
getGpsPoi(SpicePoiRequest) : SpicePoiResponse
```

The `SpicePoiRequest` object contains the central point and the radius of the search area and the category of points of interest to be searched. The `SpicePoiResponse` object contains a set of points of interest in terms of name, position, civil address and phone number.

- **getMap:** this method returns the URL of a geographical map image. The signature of the method is:

```
getMap(SpiceMapRequest) : SpiceMapResponse
```

The `SpiceMapRequest` object contains the central point of the map (latitude and longitude), map height and width in pixels and a zoom factor. The `SpiceMapResponse` object contains the URL of the map image (in JPG format).

- **getPoiMap:** this method provides a geographical map, which contains plotted points of interest. The method signature is:

```
getPoiMap(SpiceMapRequest) : SpiceMapResponse
```

The `SpiceMapRequest` object contains the central point of the map (latitude and longitude), map height and width in pixels and a zoom factor. In addition, the POI category and the radius of the search area are indicated. The `SpiceMapResponse` object has the structure described in the previous method.

- **getPtsMap:** this method returns a geographical map image containing plotted points indicated by the requester.

The signature of the method is:

```
getPtsMap(SpiceMapRequest) : SpiceMapResponse
```

The `SpiceMapRequest` object contains the central point of the map (latitude and longitude), map height and width in pixels and a zoom factor. Besides that, this object contains the list of points selected by the requester (as a list of latitude, longitude and label attributes). The `SpiceMapResponse` object has the same structure described in the previous method.

2.5 Presence Enabler

The Presence Enabler introduced in SPICE conforms to the ParlayX Presence Consumer interface as described in [3]. It contains watcher methods for requesting and subscribing presence data

The features of the Presence Enabler are the following:

- **subscribePresence:** with this method the watcher requests permission to obtain presence information from a certain user or a group of users. The watcher will not receive information on other requests unless the user or all the users involved have actually given permission. The signature of the method is:

```
subscribePresence(Presentity, Attributes, Application, Reference)
```

The `Presentity` object contains a presentity or a group of presentities whose attributes the watcher wants to monitor. The `Attributes` object gives the attribute types the watcher wants to access. (The same attribute types for all the group members). The `Application` object identifies the application the watcher needs the data for. The `Reference` object indicates the notification interface.

- **getUserPresence:** this method returns the aggregated presence data of a presentity. Only the attributes that the watcher is entitled to see will be returned. The signature of the method is:



`getUserPresence (Presentity, Attributes): Result`

The `Presentity` object contains one presentity or a group of presentities whose attributes the watcher wants to see. The `Attributes` object gives the attribute types the watcher wants to see. The `Result` object contains a number of `PresenceAttribute` objects containing the actual presence data.

3 Exposure mechanisms

3.1 Introduction

Dynamic service adaptation is a key feature of the SPICE platform. The Broker [4] already provides an adaptation mechanism based on service composition. Providing service adaptation can be done differently by actually changing the services themselves. The current section studies this alternative to the broker-based adaptation.

3.2 Exposure mechanism

Exposure is a SPICE mechanism, which introduces service components to their users, which can be end users or other components. Exposure in the SPICE platform is provided as an interaction of multiple system entities.

- The Discovery Facility allows discovery of service component meta-information, including the component access mechanisms.
- The Policy Enforcement Point (PEP) controls access to the service component.
- The Broker provides integrated service selection and access to the components from the end user and from other components considering service meta-data, service composition information, SLAs, subscription information, etc.

The Broker already supports dynamic exposure by adapting the bindings among components. Thus, a component using another component depends on the Broker to create the binding between the two components. The broker is able to consider environmental conditions, also called context or knowledge information, which is provided by the Knowledge Management Framework, KMF. In addition, the Broker can create a binding dynamically by selecting one component from a set of components that best fit the environment. The result from the end user point of view is that the service dynamically adapts to the environment.

An alternative approach to the dynamic adaptation problem is to adapt the service interface and/or meta-data according to environmental information. This is a broker-less story where the end user or component using a service component always uses the same semantic query expression. Instead of making the broker responsible for generating the semantic query expressions that contain the environmental information and for selecting the downstream services accordingly, the service interface and meta-information themselves adapt to environmental conditions. The result is that depending on the context, the same semantic query expression will select (a) different component(s) because the service interface and meta-information changes according to the context. The adaptation is done by the exposure layer that consumes relevant KMF information and exposes different service interface or meta-information depending on context. Figure 3 and Figure 4 demonstrate the difference between the two approaches.

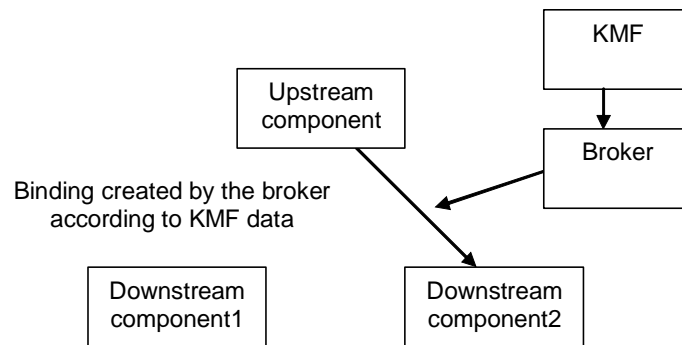


Figure 3: Broker-based component binding

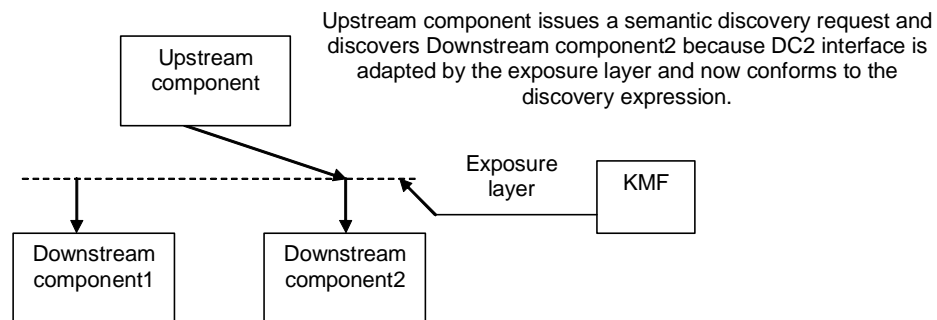


Figure 4: Broker-less service adaptation

Another aspect of service adaptation occurs when the exposure layer tries to "intelligently" add adaptors to the service, to make it comply with the requirements of the service consumer. The simplest case is the adaptation to different transport mechanisms, e.g. when a component offers a standard web service interface and the service consumer would like to access this component using CORBA. The exposure layer can detect this situation and can add a CORBA adaptor in front of the original component. In a more complicated version of this use case, the component can be semantically adapted. For example, the component requires the temperature in Kelvin but the consumer provides it in Celsius [5]. In this case, the exposure layer can insert a Celsius-Kelvin adaptor preceding the component.¹

The next sections will discuss alternatives for the broker-less setup.

¹ Note that the same functionality can be provided with the broker-based solution too. The difference is that in case of the broker-based solution, the entire adaptation logic is concentrated in the broker that does the binding according to composition descriptors – high-level description of the adaptation strategy. In case of the broker-less approach, the adaptation functionality is dispersed between the component consumers – which still have to provide semantic query expressions, although simpler ones – and the exposure layer that adapts the service interface and meta-information accordingly.

3.2.1 Adapting service interfaces and meta-data

Adaptation of service interfaces and meta-data comprises the following steps:

- Publishing service meta-data in a format that describes the relationship between environmental conditions (context variables) and service meta-data alternatives. This essentially means that multiple versions of available service meta-data are tied to conditions on knowledge elements obtained from the KMF. The exposure layer registers listeners (knowledge sinks in KMF parlance) for the knowledge elements that may affect selections among meta-info alternatives.
- KMF notifies the exposure layer that a knowledge element relevant for the selection of meta-info alternatives changed. The exposure layer then filters the KMF data to retrieve the KMF data part on which the selection is based (KMF Knowledge Sources normally produce RDF documents containing a set of KMF data). Afterwards, the selection is made.
- Next, the exposure layer republishes the service meta-info in the Discovery Facility according to the result of the selection.

An initial, simplified architecture of this service meta-information adaptation system is shown in Figure 5.

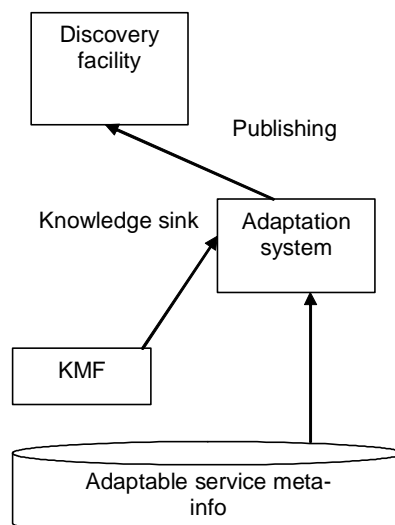


Figure 5: Simplified architecture of the service meta-info adaptation system

The algorithm above is inferior in performance compared to the broker-based approach because it adapts the service meta-data independently of whether this meta-data is accessed or not. The broker-based approach includes the relevant environment information in the semantic query, so it is irrelevant how many times this information changed since the last query. In addition, adaptation in the case of broker-based adaptation is application-dependent (only the knowledge elements relevant for the application are included in the query), while broker-less adaptation must consider each knowledge element when adapting the service meta-info.

Optimization of the simple algorithm above is possible and is necessary to achieve reasonable performance. For example, the exposure layer mechanism may cache knowledge element changes when they occur and the service meta-info may be adapted only when a Discovery Facility matcher accesses the service meta-information set. This type of optimization assumes a strong coupling between the exposure layer and the Discovery Facility. This means that the simple architecture in Figure 5 is replaced by a more complicated one where the meta-info adaptation system is strongly integrated in the Discovery Facility.

The principles of describing the service meta-data adaptation are the following:

- The adapted service meta-info files are generated from a set of meta-info file segments.
- The names of the segments and the rules controlling how the segments are combined into a set of adapted meta-info files are described in an adaptation control file.
- The adaptation control file also contains the knowledge element URIs that are used in the rules controlling the adaptation.
- There is no restriction on the format of the service meta-info files, other than that they are in XML format.

The adaptation control file is an XML file having the following sections.

- Names of the meta-info file segments used by this adaptation control file.
- Description packages of KMF elements used by this adaptation control file. One such description package contains a set of filtered knowledge element descriptions. One filtered knowledge element description contains:
 - URI of the knowledge source.
 - XSLT script used to filter the relevant knowledge element data part from the RDF document returned by the knowledge source.
 - Name of the knowledge element data part that is used to reference the value of the knowledge element data part in the adaptation control file.
- Adaptation sections. Each adaptation section generates one adapted service meta-info file. An adaptation section contains:
 - Name of the target service meta-info file to generate.
 - List of meta-info file segments that are written sequentially in the target service meta-info file in the order in which they appear in the adaptation section.
 - Conditional blocks that refer to knowledge element data part names as defined in the previous bullet point and use them in logical expressions by using some sort of expression language. Conditional blocks can be nested.

From the point of view of the users of the Discovery Facility, the result is a service whose service meta-data changes according to environmental information. This means that the

same semantic query may select the service in a certain context and may not select in another context.

3.2.2 Intelligent adaptor insertion

"Service adaptation" is an overloaded term, in that there are many different interpretations of the task to solve. Simpler approaches concentrate on the incompatibilities introduced by programming languages or middleware. For example [7] concentrates on resolving differences in type names that can be easily mapped, e.g. an IDL long and a Java int.

This document proposes the intelligent adaptor insertion problem as a "service adaptation". Given an upstream component with meta-info set A² and downstream component with meta-info set B, we look for the Service Component Network (SCN) that conforms to A on the input side and B on the output side. By inserting SCN between A and B, A is able to use the services of B, even though they are not semantically compatible. The task is illustrated in Figure 6.

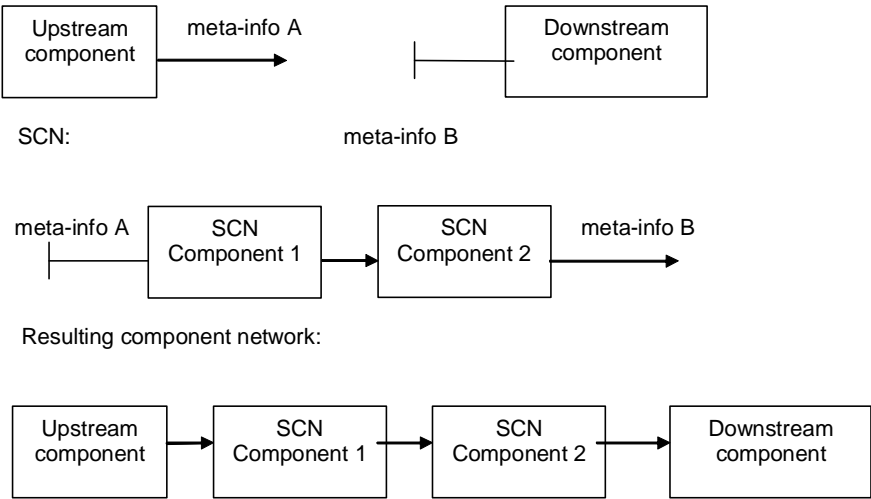


Figure 6: The adaptor insertion problem

The complexity of this task generally depends on the preconditions. In the worst case, given a set of N components, the resulting component network includes all N components. The task is then equivalent in complexity with the traveling salesman problem, which is known to be NP-complete, and which can be only solved using an unrealistic amount of computing power. Worse, unlike in the case of the traveling salesman, the same "city" (service component) can be "visited" (inserted into the chain) more than once.

If the task is simplified, e.g. the number of adaptors in SCN is restricted (e.g. to 1), the task becomes solvable in polynomial time but the "intelligence" of the solution is significantly reduced. The key question of the adaptor insertion feature is what sacrifices are made in "intelligence" in favor of performance.

² The meta-info set contains the service interface descriptions as well.

By considering the SPICE requirements, the following approach was chosen.

- The number of nodes in SCN is restricted to 1.
- Adaptor components are tagged with special meta-info that places them logically in a different directory. The adaptation system only looks for components tagged with this special meta-info (in the "adaptor directory").

Then the complete algorithm for evaluating a semantic query involving adaptor insertion is the following:

- The semantic query is sent to the directory and the result list is returned.
- If there are any results that are not of the adaptor type, the adaptor-type entries are filtered out and only non-adaptor type results are returned. In this case, there is no adaptor insertion.
- If there are only adaptors in the result set, semantic queries are reissued with the meta-data of each of the adaptors' output interface. This leads to N additional semantic queries, where N is the number of adaptors in the original result set.
- Each result set of the queries mentioned in the previous bullet point are analyzed. Adaptors are discarded from this result set (maximum number of adaptors preceding the business components is 1). If there are adaptor-component combinations that satisfy the original query (adaptor satisfies the original query and there is a business component satisfying the search expression of the adaptor's output interface), then a binding is created between the adaptor and the business component and the adaptor is returned in the result set of the original query. Note that this process of generating adaptor-business component pairs is actually broker-based. As it is not known just by looking at the adaptor whether a path over a certain adaptor will lead to a business component, this needs to be resolved before the adaptor is returned in the result set of the original query.
- The output interface of the adaptor is annotated with a query expression template. This template is used to generate second-level queries for business components. In order to enable the creation of generic adaptors (like CORBA-Web Services converter [6]), these query expression templates can take parts of the original query expressions but need to refine/change the original query expressions with the functionality provided by the adaptor.
- When binding the adaptors and their corresponding business components, the meta-information of the input interface of the business component is configured with the adaptor. This allows generic adaptors to adapt to the business component.
- As mentioned in [2], the SOA model differs from the general component model in that components are singletons in the SOA model, as they cannot be instantiated. Considering this restriction, adaptor-business component bindings are created in a session-like fashion.

The resolution process is depicted in Figure 7.

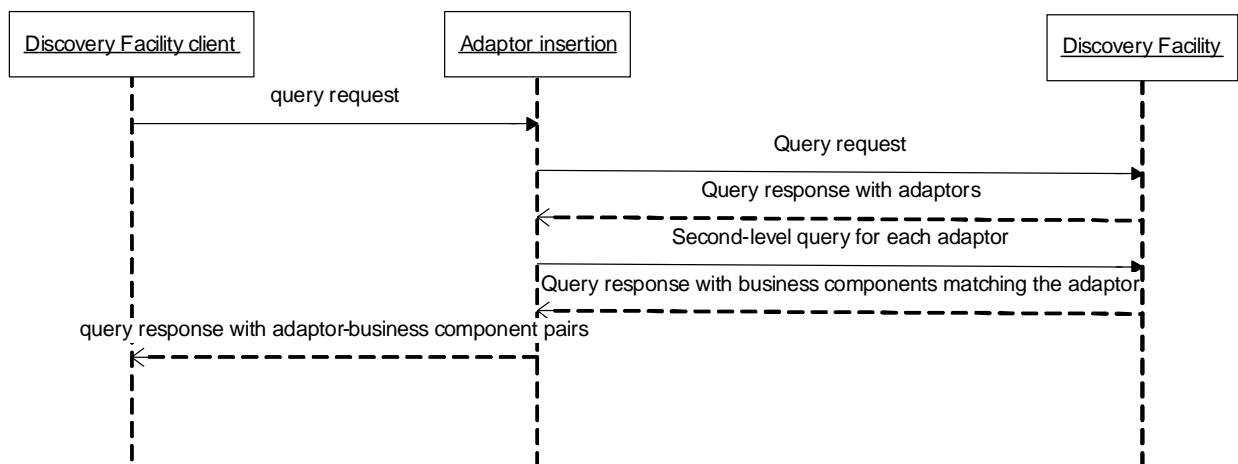


Figure 7: Interaction of adaptor insertion subsystem and the Discovery Facility

Although the adaptor insertion subsystem can be implemented as a clean proxy for the Discovery Facility, integration of this functionality into the Discovery Facility has the advantage that no discovery facility client can accidentally access the Discovery Facility directly, without going through the adaptor insertion subsystem.

3.3 Conclusion

The discussion of the exposure layer presented an alternative approach to the adaptive services presented in [4]. This alternative approach is more similar to the traditional web services mindset where the upstream component has complete responsibility of looking up downstream component references in the service directory and invoking them. In this rather rigid model, there is just one place to hide the adaptation logic: in the discovery mechanism. Hence, the idea about adapting service meta-data (including service interfaces) and dynamically inserting additional adaptor components into the service set returned by the service directory.

The advantage of this approach is that it is more natural to service component developers and it fits better into legacy web services systems. The disadvantage, however, is that the adaptation subsystem is only aware of one query expression at a time. Making general decisions (related to several adaptation decisions in the service component networks) is difficult, as in most of the cases only local decisions (concerning the actual query) can be made. The approach by default does not consider the fact that all these adaptation decisions are made in the context of an application. An application is made of components, but still has distinctive context that can affect how a component adapts to an environmental change.

The conclusion of this chapter is that broker-based adaptation has considerable advantage over broker-less adaptation because it is easy to implement application-awareness and overall adaptation decisions into broker-based adaptation systems. Broker-less adaptation systems are able to turn existing non-adaptive applications into adaptive ones more easily. If this latter consideration is important, adaptation features can be best integrated in the Discovery Facility.

4 Impact on the SPICE platform

4.1 Introduction

This chapter describes what the SPICE platform can or needs to support in order to allow third-party services to access SPICE components. First, the requirements are described that result from Access control. Then, an option is described where SPICE operators want to have more detailed statistics about the third-party access than provided by the accessed component.

4.2 Access Control

As control of third-party access is part of the broader SPICE access control system, this section is related to a number of other SPICE deliverables. An early illustration of the SPICE access control framework components [8] introduces and defines a set of initial concepts and functions such as the Policy Enforcement Points (PEPs), Policy Decision Points (PDPs), Authentication and Authorization Support (AAS), Identity Provider (IdP), Service Level Agreements (SLAs), billing, User Privacy Management (UPM), etc. Note that PEPs are defined on different levels, such as the SPICE Service Execution Environment, (SEE), the Service Component (SC) and the protocol layer. The initial architecture specification for service access control [9] describes the elements and functions in detail. The final concepts and specifications for service access control [10] finalize the architecture. The Cookbook [2] contains a chapter on authentication and authorization mechanisms, which describe the subscription concept for users and SCs, the authentication concept (by certificate, via IdP and via IMS), the authorization concept, the session concept for authentication purposes, the structure of SC metadata for access control, and policy decision and enforcement for basic and XACML-encoded policies. The Broker [3] is also related to access control; although primarily the “broker simplifies the user's life in accessing services”, it also behaves as a PEP when obtaining subscription information from the Subscription Manager and post-filtering the component list returned by the Discovery Facility, so that components that the user is not subscribed to are not selected during composition.

As can be seen from the above list of related deliverables, the SPICE access control architecture is pretty complex. Therefore, sections 4.2.1 to 4.2.5 give a brief overview of the main elements of access control, resulting in a simplified conceptual framework from which the current topic — third-party access control — can be easily understood (note that this overview is very simplified, and the above-listed deliverables need to be consulted for the details). The requirements that third parties have to fulfill in order to be compliant with the SPICE access control system are described in chapter 5, section 5.2.

4.2.1 Main Concepts

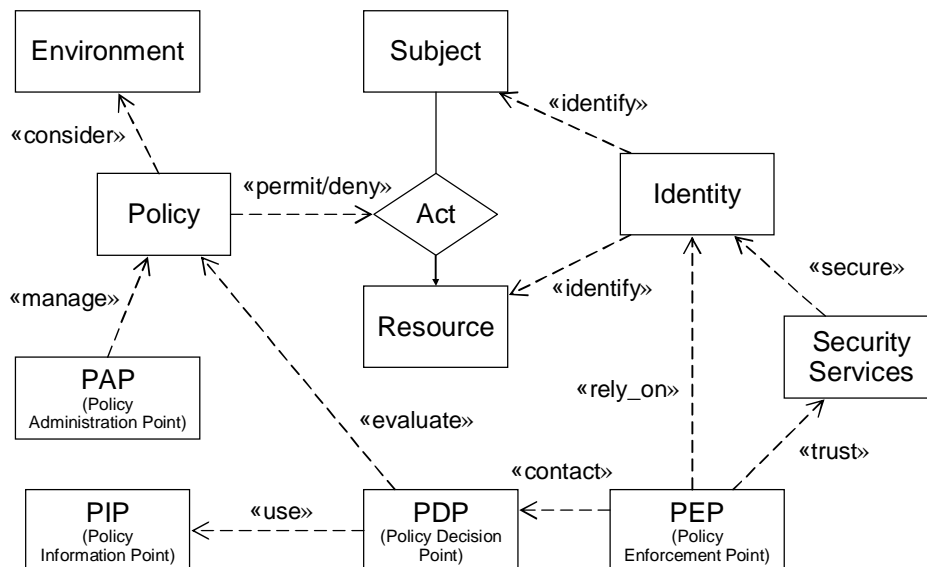


Figure 8: Main concepts of access control

Figure 8 depicts the main concepts of access control by means of UML classes, associations and dependencies. The policy-related notions are borrowed from XACML [1]. The figure reads as follows. The task is to control the Actions of Subjects on Resources. A Subject may be a user, a SPICE service component (SC), a third-party component, etc. An Action is e.g. calling a method of a SC. A Resource may be a SC, user data, etc. in SPICE. A Policy declares who (Subject) may do what (Action) on what (Resource) under what circumstances (conditions, Environment). Policies are managed — created, deleted, modified — via a Policy Administration Point. The actual access control is performed by a Policy Enforcement Point (PEP) by enforcing authorization decisions; a PEP resides in the messaging path and is able to block messages. A PEP typically contacts a Policy Decision Point (PDP) for making an authorization decision; the PEP formulates a decision request, passes it on to the PDP, which then evaluates the applicable policy/policies and renders an authorization decision. The PDP (or alternatively the PEP) may use an additional entity called Policy Information Point (PIP) for retrieving additional information needed for the decision; an example of a PIP is the Subscription Manager, which is able to return information on who is subscribed to a given service or SC. When dealing with Subjects and Resources, a PEP relies on Identities that are secured by Security Services (cf. section 4.2.3). The entity implementing the Security Services is a trusted entity for a PEP. A more detailed description of the concepts can be found in deliverables D6.2 [8], D6.3 [9] and D6.4 [10].

4.2.2 Trust Model

The trust model between SPICE platforms [10] is based on the following principle: communication between a SPICE platform at operator A and a SPICE platform at operator B is possible only if A and B mutually *trust* each other. Trust is defined as the validity of the following two assumptions:

- The communication between the parties is *secure*. In this context, Security of communication means (1) authenticity of data origin a.k.a. identity, (2) data integrity, (3) confidentiality (optional) and (4) anti-replay protection (optional). Communication is secured by technical measures as described later (section 4.2.3 on Security Services).
- The other party is *friendly* i.e. it is not assumed to do malicious actions. Friendliness can be ensured by means of legal measures (written contract between A and B, similar to roaming agreements), or by technical measures, which are both beyond the scope of this document.

When applying the above principle to the case of a SPICE platform at operator A and any service component at another party B, we get an asymmetric version of the principle: the service component at party B may access the SPICE platform at operator A only if A *trusts* B, where trust is defined as above (considered only from A's point of view). This principle will later be extended to also allow (limited) access from untrusted parties (section 5.2.1).

Note that the above definition of trust is quite limited in scope, but it serves our purpose in SPICE.

4.2.3 Security Services

Security services ensure the security of communication between parties (of a different trust domain). They correspond to the four aspects of secure communication listed in the previous section, namely:

- (1) authentication of data origin a.k.a. identity,
- (2) data integrity,
- (3) confidentiality (optional) and
- (4) anti-replay protection (optional).

Security services are implemented separately from the rest of the platform elements. This relieves service and component developers (even the developers of PEPs, PDPs, Broker, etc.) from the difficult task of implementing security features, and it makes the platform easier to maintain and configure. The elements implementing the security services are called Security Gateways (SEGs), like in the 3GPP specification of IP network layer security [11],

Security Gateways (SEGs) in SPICE possess the following characteristics:

- They reside on the border of platform operator networks with all inter-platform SPICE traffic passing through them. Note that user-platform traffic flows via different channels (cf. AAS in the SPICE access control architecture).
- They provide the necessary security services (1-4 as above), i.e., a pair of SEGs, at operator A and B, respectively, establish a point-to-point secure channel between operators A and B.

- At each operator, there is one logical SEG per foreign operator. This allows clean and easy configuration of SEGs, in accordance with the pair-wise legal agreements between the platform operators.
- SEGs are transparent to the upper layers.

More details on SEGs can be found in deliverable D6.4 [10].

4.2.4 Identification

In general, subjects (users, SCs) and resources (SCs, data) are identified by URIs. The URIs should be unique to avoid ambiguities.

In addition, identities are also used when an SC is informed about its requester(s). In general, as an SC may be invoked as part of a composite service, there is a chain of requesters. Assuming HTTP transport for component method invocation, the identity chain is transferred in a dedicated HTTP header called X-SPICE-Asserted-Identities.

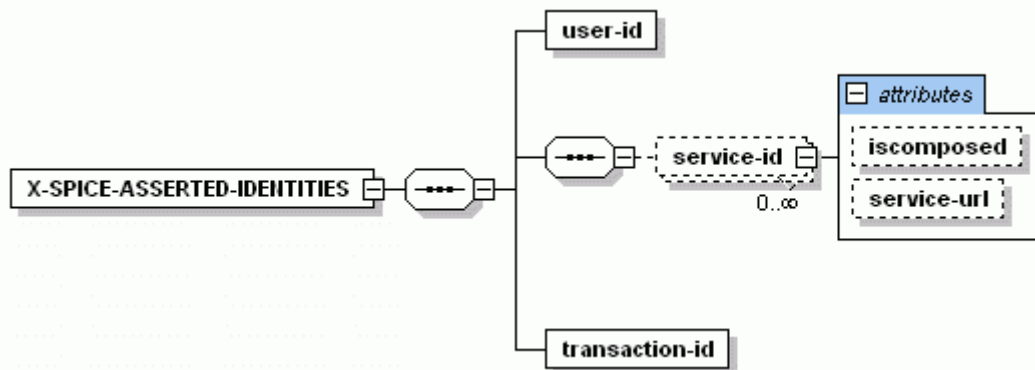


Figure 9: X-SPICE-Asserted-Identities HTTP request header syntax

Figure 9 illustrates the syntax of the X-SPICE-Asserted-Identities HTTP request header. Each component has to add its own identity whenever it invokes another component.

4.2.5 Policies

According to the Cookbook [2], basic access policies, such as when to start a session, or which services are open for public access, are written in a proprietary form, whereas more complex policies are written in XACML. More details on access policies can be found in deliverable D6.4 [10].

4.3 Additional statistics

The enabler interfaces in chapter 2 show that some enablers are of the ‘single query’ type, as opposed to the ‘components with callback’ type. Components of the ‘single query’ type are invoked and return a result. Components with a callback may return immediately with a result, but they also send invocations to the invoking entity. The instant messaging enabler has a callback for receiving messages.

It is possible that SPICE operators want to know exactly how often services are invoked. Enablers with callback have another statistic that may be important for SPICE operators, namely the number of invocations to the external component. For these kinds of cases it

may be necessary, if the enabler itself does not provide these statistics, to provide a cookbook compliant front-end component in front of the enabler. Figure 10 shows which interfaces are involved.

external interface

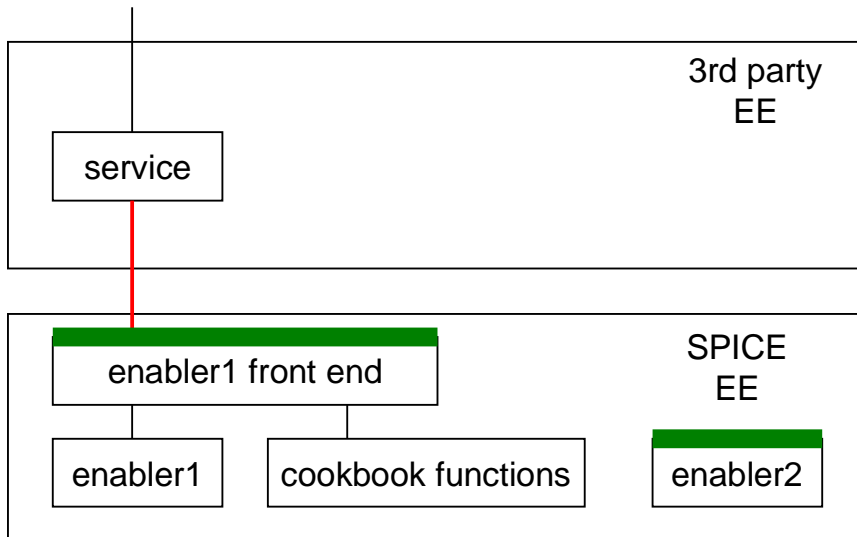


Figure 10: SPICE enabler with front-end component

The front-end component is an enabler specific component that intercepts all invocations in both directions, and maintains statistics about the number of invocations. The statistics are passed to the cookbook functions Component Supervisor and Performance Manager. If the SPICE operator prefers the interface to the external party to be different from the real interface to the enabler, this may be implemented in the front end component as well.

5 SPICE impact on the component

5.1 Introduction

This chapter describes which interfaces need to be supported by the third-party service or by the component in that service that invokes the SPICE component. First, the requirements are described that result from Access control. Then, the relevant part of the interface to the Service Broker is provided.

5.2 Requirements for Third-Party Access

5.2.1 Trust Model

Regarding third-party access to a SPICE platform, we distinguish between the following two cases:

- *Trusted third-party access*, when the requester is a third-party trusted by the platform operator (i.e. with whom also a written contract is made; this is similar to inter-SPICE access).
- *Public access*, when the requester is an untrusted entity.

Depending on which of the above cases a third-party belongs to, it may have access to any number of services, controlled by policies via PEPs.

5.2.2 Security Services

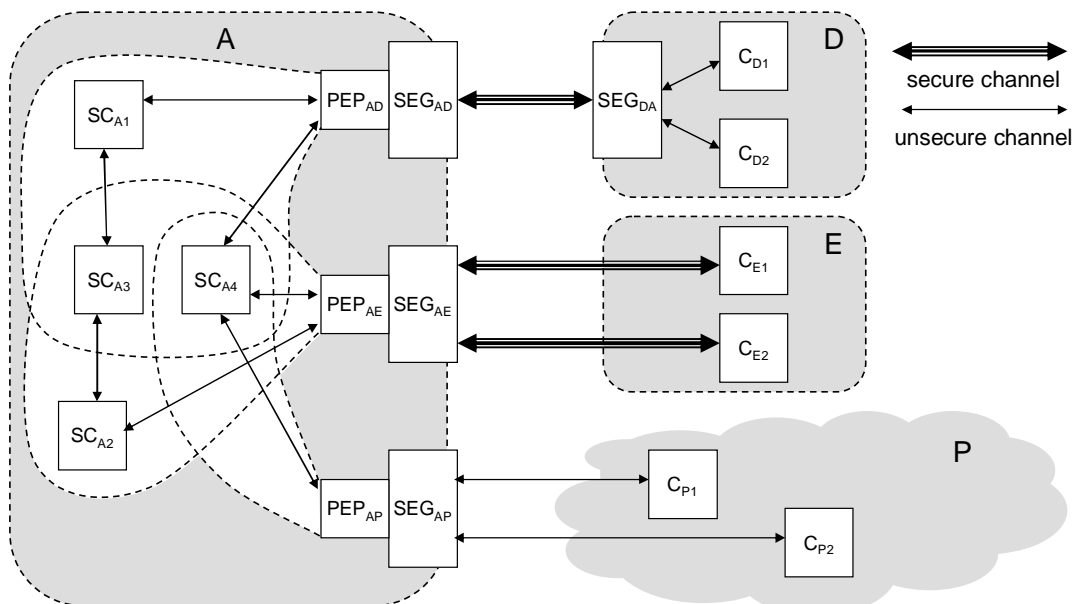


Figure 11: Security Gateways (SEGs), trusted third parties and public access

Figure 11 illustrates the cases listed in the previous section and the associated security services. Operator A opens up part of its platform for a trusted third-party D by means of a dedicated SEG and PEP; party D deploys a counterpart SEG on the border of its domain.

Operator A similarly opens up another part of its platform for the trusted third-party E. The E party does not deploy a SEG but lets the individual components implement the necessary security services required by the corresponding SEG at operator A. Furthermore, operator A also opens up a small part of its platform for public access by means of a dedicated SEG and PEP; the platform part opened up for public access typically falls into the intersection of the parts opened for trusted partners, as also illustrated in the Figure 11 (SC_{A4}).

To summarize, trusted third parties are required to implement the security services mandated by the SPICE platform operator.

5.2.3 Identification

Trusted third parties are required to use (subject and resource) identifiers that are compliant with the SPICE specification. At the current level of detail, SPICE identifiers are URIs and unique. As a special requirement, the chain of requester identities should be passed to SCs in case they need to be invoked as part of a composite service. More details on SPICE identities can be found in deliverable D6.4 [10].

5.2.4 Policies

The trusted third parties are required to provide all necessary information for the SPICE platform operator, to carry out access control decisions. More specifically:

- The identity and attributes of the accessing third-party components must be made available to the SPICE platform. The identity of third-party accessors may also be necessary for the SPICE Subscription Manager. The necessary attributes of third-party accessors can be transferred to the SPICE platform (under a PIP) or can be made available through a PIP service endpoint at the third-party.
- If access to the third-party component is controlled by PEPs in the SPICE platform, then the relevant policies must be available for the PDPs in the SPICE platform, either by deploying them to the SPICE platform or by making them available via a service endpoint at the third-party.

5.2.5 Conclusion

From an access control point of view, trusted third-party access is not much different from inter-SPICE access, and “untrusted third-party access” is equivalent with public access. A small number of high-level requirements for trusted third parties have been formulated. Based on these requirements, and after consulting the referenced SPICE deliverable D6.4 [10] for the details, third-party providers are able to start designing and implementing the required security machinery. Note that this effort is anticipated to require further discussions with the affected SPICE platform operator in order to nail down some important details. Such details may concern security associations to be used by the SEGs, operator-specific requirements on service and user identifiers.

5.3 Interface to the Service Broker

The relevant operation of the interface to the Service Broker is defined as follows:

- **BindService:** This operation attempts to find a service that matches the provided pattern complemented with input received from the provided Knowledge Source. For that purpose it queries the Knowledge Resource and includes the result in the pattern with which it queries the Discovery Facility. When the query to the Discovery Facility is successful, a reference to a service endpoint is returned. The signature of the operation is the following:

```
BindService(DynamicEndpoint Input , String SessionId) :  
    EndpointReference
```

The `Input` parameter contains a Knowledge Source Reference and a pattern. The Knowledge Source Reference contains a Knowledge Source URI, where the Knowledge Source can be reached, and a knowledge pattern, used in the query to the Knowledge source. The result of the query is used in the original received pattern to query the Discovery Facility. The result of the query to the Discovery Facility, a service URI, is returned in the `EndpointReference` that points to the service.

6 Conclusion and outlook

In order to allow third-party access, a number of new mechanisms have been investigated, and extensions of existing mechanisms have been considered.

In order to maximize the number of third-party service requests to the Discovery Facility that find a match, a new, broker-less dynamic adaptation mechanism has been studied. The conclusion is that the broker-based adaptation mechanism is still preferred, because it is easy to implement application-awareness and overall adaptation decisions into broker-based adaptation systems. Broker-less adaptation systems are preferred when it is necessary to turn existing non-adaptive applications into adaptive ones. If this latter consideration is important, adaptation features can be best integrated into the Discovery Facility.

For Access Control, the conclusion is that for SPICE, trusted third-party access is not much different from inter-SPICE access, and “untrusted third-party access” is equivalent with public access. A number of high-level requirements for trusted third-party access have been formulated. A number of these requirements are covered by the introduction of a Security Gateway in the SPICE platform. More detailed requirements can be found in [10] in which the final concepts and specifications for service access control have been described.

A SPICE operator may optionally implement a service-specific front-end component, fully Cookbook compliant, to collect additional statistics about the use of the SPICE component and e.g. the number of callback invocations from the SPICE component to the service.

Apart from the access control, the third-party service also needs to implement the interface to the Service Broker, and the actual service interface.

7 References

- [1] eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard, 1 Feb 2005.
- [2] SPICE D2.1, Cookbook and Interface Specification of Service Enabler Components, December 2006.
- [3] 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Open Service Access (OSA); Parlay X Web Services; Part 14: Presence (Release 6), 3GPP TS 29.199-14 V6.7.0 (2007-06)
- [4] SPICE D2.3, Service broker architecture for service enabler access, April 2007.
- [5] Jyotishman Pathak, Doina Caragea, and Vasant G Honavar, *Ontology-Extended Component-Based Workflows: A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components*, *2nd Workshop on Semantic Web and Databases, Toronto, Canada, August 29-30, 2004*
- [6] CORBA-WSDL/SOAP Interworking,
http://www.omg.org/technology/documents/formal/CORBA_WSDL.htm
- [7] Jeffrey Hau, William Lee, Steven Newhouse, *Autonomic Service Adaptation in ICENI using Ontological Annotation*
- [8] SPICE D6.2, WP6 Component Demonstration based on Paper Mock-Ups, September 2006.
- [9] SPICE D6.3, Initial Service Access Control Architecture, January 2007.
- [10] SPICE D6.4, Final concepts and specifications for Service Access Control, December 2007.
- [11] 3GPP TS 33.210, 3G Security; Network Domain Security; IP network layer security, V7.0.0.