Unified Service Description Language (USDL) Functional Module

December 28, 2009

Edited by SAP Research

Abstract. This document describes what is called the Functional module in the third version of the Unified Service Description Language (USDL). USDL was developed as a holistic approach to describe entities provisioned into service networks, which considers and connects business, operational (functional) and technical aspects of service description. The Functional module covers concepts that describe what a service does/offers and how consumers are able to access this functionality.

Table of Contents

1 Introduction

As outlined in the central document of this series *"USDL Technical Overview Paper"*, services are becoming the backbone for electronic commerce. Especially the trend to provision IT-based services outside company "firewalls" with the help of intermediaries is on the increase, as it allows organizations to take new opportunities relatively quickly. In this context services are seen as tradable entities that constitute a well defined, encapsulated, reusable and business-aligned set of capabilities. The term business service is used for such services, in order to distinguish them from other types, e.g., those that are provided in a service-oriented IT infrastructure within an organization.

The Unified Service Description Language (USDL) defines a way to describe services from a business and operational point of view and align this with the technical perspective. While the latter is captured quite well by existing service description languages, USDL explicitly enables to express business characteristics set by an organization for the purpose of providing means for consumers to invoke and use business services and for intermediaries to (re)use and repurpose services. A detailed explanation of the scope and objectives of USDL is given in *"USDL Technical Overview Paper"*.

USDL on a whole is made up of a set of modules, each addressing different aspects of the overall service description. Modularization was introduced to improve readability of the model, which drastically grew in size compared to its predecessor. The modules have dependencies among each other (shown in Figure 1), as they may reuse concepts from other modules. Currently, there are 8 modules in the set that constitutes USDL version 3.0.

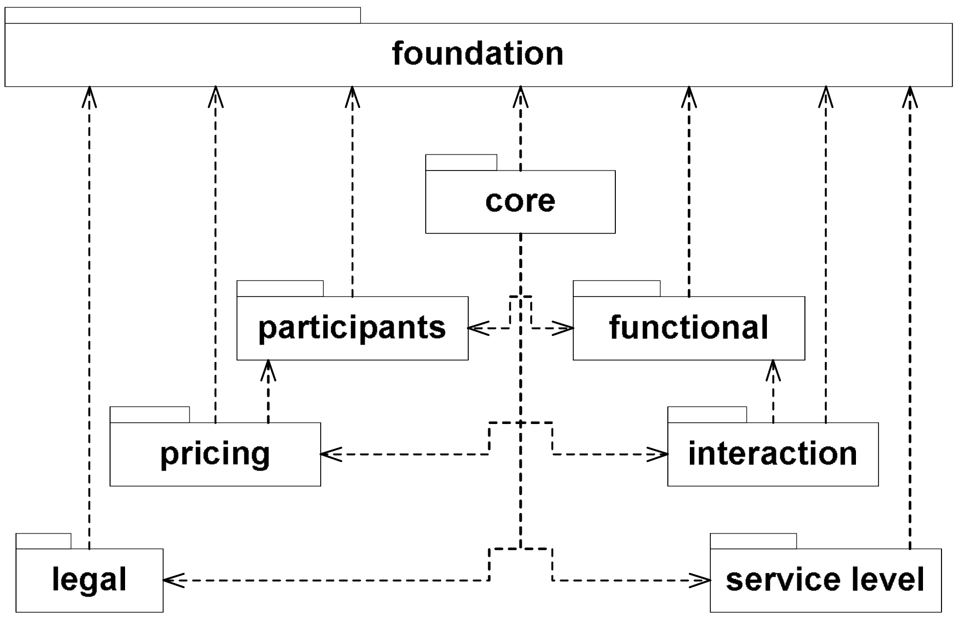

Figure 1 Packages comprising the USDL model and their dependencies (represented as arrows)

1.1 About this document

The USDL meta-model is formally defined in Ecore (the meta-modeling language of EMF), with each USDL module being captured in a separate package. This document is one in a series of USDL documents and covers the Core module defined in package “core”. The series also includes:

- *USDL Technical Overview Paper*
- Module-specific documentation of the modules *Foundation*, *Core*, *Participants*, *Interaction*, *Pricing*, *Service Level* (includes geographical and temporal availability) and *Legal*

The document only provides insights into the concepts of the Functional module. For a complete overview of USDL it is necessary to go through all documents of the series.

1.2 Acknowledgements

Work on USDL has been mainly carried out in the context of the THESEUS TEXO– a project in the frame of the THESEUS Lighthouse research program initiated by the German Ministry of Information and Technology.

Several European, German, and Australian research projects also contributed to the development of USDL. Naturally, there is an extensive number of people that have contributed to conceptualization and documentation of USDL either directly or through feedback. Rather than giving the full list of individuals, it shall suffice to name the institutions they work for. The full list is available in the *USDL Technical Overview Paper* and on http://www.internet-of-services.com.

SAP Research

European Research Center for Information Systems (ERCIS), University of Muenster

Queensland University of Technology, Brisbane, Australia

University of New South Wales, Sydney, Australia

2 Functional Module

One of the most integral parts of every service description is to express what a service is able to do and how a consumer is able to use what the service provides. While the latter is well covered by existing technical service descriptions, at least for automated services, the former is less explored. In order to equally enable the description of human and automated services the Functional module captures both, conceptual functionality on one side and the technical realization that exposes this functionality on the other. It is important to distinguish between these two concepts, one being the subject of the service itself and the other being the service’s interface. The reason is that a single service may be available, completely or in parts, over several interfaces. Interface in this context means a set of concrete technologies through which the service can be accessed. A simple example is an automated service that has a WSDL-based Web service interface and a REST interface. Even within one interface there might be different ways of access, e.g. by means of providing multiple access points that expect different sets of technical protocols to be used for communication. This concept is most prominently featured in WSDL with its ports and bindings.

Currently there are no standardized means to capture the functionality provided by a service. A common conceptualization, used for example in SoaML (OMG) or the SOA Reference Model (OASIS), is capability modeling. Oaks, Hofstede and Edmond discuss the concept of capabilities in (Oaks et al., 2003). According to them capabilities express the ability to perform a course of action that achieves a result. When viewed on a whole, this constitutes the functionality offered as *the* service. Actions themselves produce outcome, e.g., something is created, transformed, or destroyed. Action also means that an actor is involved, who/which carries out the action. Usually, actors operate on one or more objects, consuming and producing some of them, while others are only affected. It is furthermore common that actors use resources as tools to perform an action. In some cases it is necessary to describe conditions that have to hold before an action can be started, as well as the effects that set in once the action is completed.

Exposing conceptual functionality of a service, similar concepts can be observed on technical interfaces of automated services. Such interfaces consist of operations that have typed input and output parameters. Additionally, faults/exceptions may be consumed or produced during an operation. Semantic service approaches further specify pre- and post-conditions of operations in a given expression language.

2.1 Module Info

Parameters of the package that captures the module

- Namespace: *http://internet-of-services.com/usdl/modules/functional*
- Name: *functional*

The remainder of this section describes the classes and enumerations that are part of the package. Figure 2 depicts a class diagram of the package.

Note: Example fragments are provided for some of the classes. In order to improve readability they are presented in XML-based pseudo syntax. This is **NOT** the official USDL syntax, which is still under development. However, there currently exists a serialization format that is XMI-based and supported through a USDL editor developed by SAP Research.

2.2 Module Dependencies

In order to understand concepts from referenced modules in detail, it is recommended to go through the following documents describing other USDL modules:

- Foundation
- Core

A quick overview of the concepts most widely used in the Functional module is given below. This will avoid extensive jumping between documents.

Name	Type	Module	Description
IdentifiableElement	Interface EClass	Foundation	Serves as the super type of all elements of USDL that can be uniquely identified, either globally or within a certain namespace
ElementDescription	EClass	Foundation	A generic concept that provides various information elements to describe USDL concepts
Artifact	EClass	Foundation	A generic concept that allows to include links to USDL-external service metadata, as well as arbitrary documents, files, web pages, etc.
Resource	EClass	Foundation	A generic concept to represent real-world objects of various types, e.g. an application, a system, a tool used to perform a service, or an object a service is performed on
ExpressionElement	EClass	Foundation	A generic concept to model expression in an arbitrary expression language

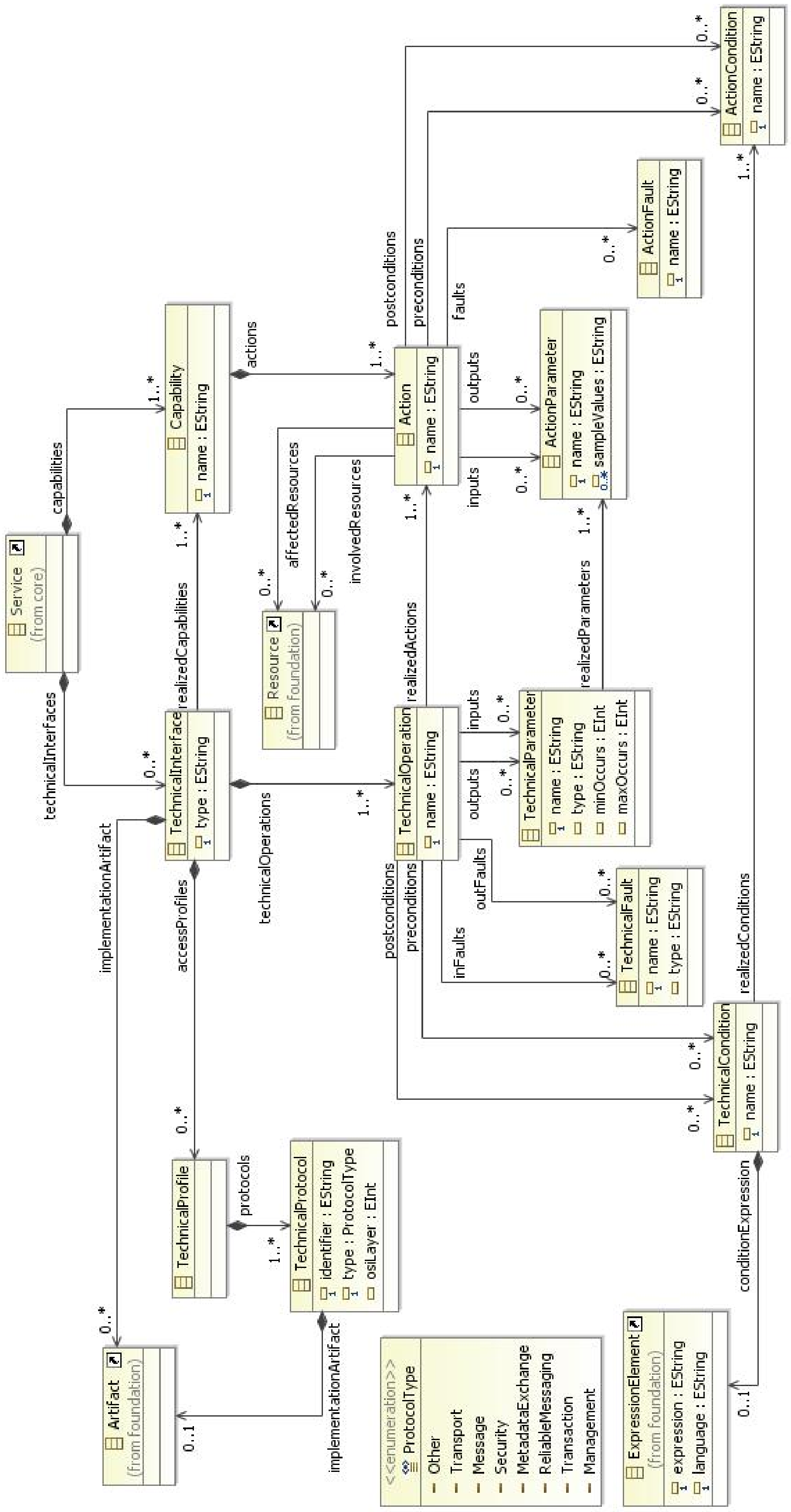

Figure 2 Class diagram of the package that captures the Functional module

2.3 Capability

Capability is used to capture an informal description of what the service does, i.e., its functionality. Capabilities express the ability to perform a course of action, which ultimately constitutes the service rendered to the consumer. Hence, a service has to have at least one capability; otherwise it cannot be considered a service.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

Capability			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the capability
Relations			
Name	Type	Cardinality	Description
actions	Action	1..*	The set of actions comprised in the capability
description	Element Description	0..1	A description of the capability
Examples (in pseudo concrete syntax)			

```
<service>
…
 <capabilities id=”cap345”>
   <name> Open Term Deposit Account </name>
   <actions> … <actions>
   <actions> … <actions>
   <description>
     <longDescription>
       Consumers are able to open a term deposit account for 3, 6, or 12 months.
     </longDescription>
   </description>
 </capabilities>

 </capabilities>
…
</service>
```

2.4 Action

Action is used to capture an informal description of individual functions/operations performed as part of the service. An action sometimes operates on resources, i.e., transforms a resource in terms of appearance, state, etc. It might also involve other resources, which are utilized to carry out the action. Actions might take inputs in order to be performed and might produce outputs as a result of them being performed.

Example 1: Project Management

As part of project management the action "create project time plan" is performed. The action produces the detailed time plan of the project (output) taking into account (input) parameters like project goals, activities / work items, available resources (workforce, budget, ...), or project duration.

Example 2: Sculpturing

A sculptor creates a marble sculpture (action). He has a concept drawing of the sculpture (input). According to this drawing, he uses his tools (involved resources) to work on a block of marble (manipulated resource). In the end he will have produced the sculpture (output).

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

Action			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the action
Relations			
Name	Type	Cardinality	Description
inputs	Action Parameter	0..*	The set of input parameters required for performing the action
outputs	Action Parameter	0..*	The set of output parameters produced by performing the action
faults	ActionFault	0..*	The set of faults that may occur during performing the action
preconditions	Action Condition	0..*	The set of conditions that have to be satisfied before the action can be performed
postconditions	Action Condition	0..*	The set of conditions that hold after the action is completed successfully
affectedResources	Resource	0..*	The set of resources transformed/manipulated by the action
involvedResources	Resource	0..*	The set of resources utilized to perform the action
description	Element Description	0..1	A description of the action
Examples (in pseudo concrete syntax)			

```
<service>
…
 <capabilities id="cap345">
  …
  <actions id="action433">
   <name> Collect Personal Details </name>
   <description>
    <longDescription>
     The first step of application is to collect personal details from the customer.
    </longDescription>
   </description>
   <outputs> #param123 </outputs>
  </actions>
  …
  <actions id="action435">
   <name> Open Account </name>
   <description>
    <longDescription>
     With all details collected, the account can be created and provisioned.
    </longDescription>
   </description>
```

(continues on next page)

```
    <inputs> #param123 </inputs>
    <inputs> #param124 </inputs>
    <outputs> #param127 </outputs>
   </actions>
   …
  </capabilities>
…
</service>
```

2.5 ActionParameter

ActionParameter is used to capture input to and output of an action. Parameters can be something very vague like an idea. On the other hand they can be something specific, such as the architecture blueprint of a building.

- Ecore Type: EClass
- Interfaces: ???
- Superclass: N/A

ActionParameter			
Attributes			
Name	**Type**	**Cardinality**	**Description**
name	EString	1	The name of the action parameter
sampleValues	EString	0..*	An optional list of sample values given in an informal description
Relations			
Name	**Type**	**Cardinality**	**Description**
description	Element Description	0..1	A description of the action parameter
Examples (in pseudo concrete syntax)			

```
<actionParameter id=”param123”>
  <name> Personal Details </name>
  <description>
    <longDescription> Personal details of a person </longDescription>
  </description>
</actionParameter>
```

2.6 ActionCondition

ActionCondition is used to capture information about state of objects (abstract objects, resources, etc.) that has to be reached before something can happen or after something has happened.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

ActionCondition			
Attributes			
Name	**Type**	**Cardinality**	**Description**
name	EString	1	The name of the action condition

Relations			
Name	Type	Cardinality	Description
description	Element Description	0..1	A description of the action condition
Examples (in pseudo concrete syntax)			

2.7 ActionFault

ActionFault is used to capture information about faults/exceptions that may occur when an action is performed.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

ActionFault			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the action fault
Relations			
Name	Type	Cardinality	Description
description	Element Description	0..1	A description of the action fault
Examples (in pseudo concrete syntax)			

2.8 TechnicalInterface

A **TechnicalInterface** is exposed by automated services, i.e. those with a technical (software) implementation. Services may offer several technical interfaces realized in different technologies. Each of the interfaces may provide all or part of the service's functionality (set of capabilities).

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalInterface			
Attributes			
Name	Type	Cardinality	Description
type	EString	mandatory	The identifier of the technology used to implement/realize the interface; ***example:*** *http://schemas.xmlsoap.org/wsdl/ (WSDL 1.1 namespace)*
Relations			
Name	Type	Cardinality	Description
technicalOperations	Technical Operation	1..*	The set of operations exposed by the interface
accessProfiles	Technical Profile	0..*	The set of profiles that describe how to access the interface on a protocol level

implementationArtifact	Artifact	0..*	Link to the file(s)/resource(s) that contains the formal definition of the technical interface; ***example:*** *link to a WSDL document*
realizedCapabilities	Capability	1..*	A technical interface exposes, i.e. realizes access, to at least one capability of the service
description	Element Description	0..1	A description of the technical interface

Examples (in pseudo concrete syntax)

```
<service>
…
 <technicalInterface>
  <type> http://schemas.xmlsoap.org/wsdl/ </type>
  <description>
   <name> SOAP-based WSDL interface </name>
  </description>
  <accessProfiles> … </accessProfiles>
  <technicalOperations> … </technicalOperations>
  <realizedCapabilities> #cap345 </realizedCapabilities>
  <implementationArtifact>
   <type> TechnicalMetadata </type>
   <source> service provider </source>
   <mimeType> application/xml </mimeType>
   <uri> http://services.moonbank.com/services/account/P05529-RSM-DL006.wsdl </uri>
   <description>
    <name> WSDL interface </name>
   </description>
  </implementationArtifact>
 </technicalInterface>
…
</service>
```

2.9 TechnicalOperation

A **TechnicalOperation** constitutes a distinct set of functionality exposed through a technical interface. Invoking a technical operation means that a number of the actions provided by the service are performed.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalOperation			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the technical operation
Relations			
Name	Type	Cardinality	Description
inputs	Technical Parameter	0..*	The set of input parameters that the operation expects
outputs	Technical Parameter	0..*	The set of output parameters that the operation returns
infaults	Technical Fault	0..*	The set of faults that can be processed with the operation

outfaults	Technical Fault	0..*	The set of faults that may be produced by executing the operation
preconditions	Technical Condition	0..*	The set of conditions that have to be satisfied before the operation can be invoked
postconditions	Technical Condition	0..*	The set of conditions that hold after the operation completed successfully
realizedActions	Action	1..*	A technical operation exposes, i.e. realizes access to, at least one conceptual action
description	Element Description	0..1	A description of the technical operation
Examples (in pseudo concrete syntax)			

```
<service>
…
 <technicalInterface>
 …
  <technicalOperations>
   <name> OpenTermDepositAccount </name>
   <inputs> #techParam765 <inputs>
   <realizedActions> #action435 </realizedActions>
  </technicalOperations>
 </technicalInterface>
…
</service>
```

2.10 TechnicalParameter

TechnicalParameter is used to capture the inputs to and outputs of a technical operation. Technical parameters are information objects that hold data processed and produced by an operation.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalParameter			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the technical parameter
type	EString	0..1	The identifier of the parameter's type
minOccurs	EInt	0..1	The minimum number of instances of the parameter that have to be in a request/response; **default:** 1
maxOccurs	EInt	0..1	The maximum number of instances of the parameter that may be in a request/response; **default:** 1
Relations			
Name	Type	Cardinality	Description
realizedParameters	Action Parameter	1..*	A technical parameter is a partial or full realization of one or more conceptual action parameters
description	Element Description	0..1	A description of the technical parameter

Examples (in pseudo concrete syntax)

```
<technicalParameter id="techParam765" xmlns:mbTypes=" http://servicetypes.moonbank.com/account/">
  <name> Application Details </name>
  <type> mbTypes:termDepositOpenRequest </type>
  <realizedParameters> #param123 </realizedParameters>
  <realizedParameters> #param124 </realizedParameters>
</technicalParameter>
```

2.11 TechnicalCondition

TechnicalCondition is used to capture information about technical state that has to be reached before something can happen or after something has happened.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalCondition			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the technical condition
Relations			
Name	Type	Cardinality	Description
realizedConditions	Action Condition	1..*	A technical condition is a partial or complete realization of one or more conceptual action conditions
conditionExpression	Expression Element	0..1	The condition formalized as an expression using an arbitrary expression language
description	Element Description	0..1	A description of the technical condition
Examples (in pseudo concrete syntax)			

2.12 TechnicalFault

TechnicalFault is used to capture information about technical faults/exceptions that may be processed by an operation or may occur during its execution.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalFault			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the technical fault
type	EString	0..1	The identifier of the fault's type

Relations			
Name	Type	Cardinality	Description
description	Element Description	0..1	A description of the technical fault
Examples (in pseudo concrete syntax)			

2.13 TechnicalProfile

A **TechnicalProfile** captures information how a service can be accessed technically. It mimics the concept of a binding in WSDL, which allows for a Web service to be exposed through different protocols. Similarly, a technical profile groups a number of technical protocols that are used for a specific method of access/interaction.

Example: A service offers 2 methods of access

- *SOAP over HTTP endpoint with security and reliable messaging features*
- *RESTful endpoint with different security but no reliable messaging*

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalProfile			
Relations			
Name	Type	Cardinality	Description
protocols	Technical Protocol	1..*	The set of protocols grouped by the profile
description	Element Description	0..1	A description of the technical profile
Examples (in pseudo concrete syntax)			

```
<service>
…
 <technicalInterface>
 …
  <accessProfiles>
   <description>
    <name> SOAP-based Web Service </name>
   </description>
   <protocols> … </protocols>
   <protocols> … </protocols>
  </accessProfiles>
 …
 </technicalInterface>
…
</service>
```

2.14 TechnicalProtocol

TechnicalProtocol captures basic information about standard protocols of Information Technology, which are employed by the service during communication and interaction with clients.

- Ecore Type: EClass
- Interfaces: N/A
- Superclass: N/A

TechnicalProtocol			
Attributes			
Name	Type	Cardinality	Description
identifier	EString	1	The unique identifier of the protocol as defined in the protocol's formal specification
type	ProtocolType	1	The aspect of communication / interaction with the service addressed by the protocol
osiLayer	EInt	0..1	The layer in the OSI Reference Model where the protocol is located; **valid values:** 1-7
Relations			
Name	Type	Cardinality	Description
implementationArtifact	Artifact	0..1	Optional link to the file(s)/resource(s) that contains the formal definition of communication aspect addressed by the protocol; ***example:*** *link to a WS-Policy file that defines WS-Security configurations*
description	Element Description	0..1	A description of the technical protocol
Examples (in pseudo concrete syntax)			

```
<service>
…
 <technicalInterface>
 …
  <accessProfiles>
  …
   <protocols>
    <identifier> http://schemas.xmlsoap.org/soap/envelope/ </identifier>
    <type> Message </type>
    <osiLayer> 7 </osiLayer>
    <description>
     <name> SOAP </name>
    </description>
   </protocols>
   <protocols>
    <identifier> RFC2616 </identifier>
    <type> Transport </type>
    <osiLayer> 7 </osiLayer>
    <description>
     <name> HTTP </name>
    </description>
   </protocols>
  …
  </accessProfiles>
 …
 </technicalInterface>
…
</service>
```

2.15 ProtocolType

ProtocolType indicates which aspect of communication or interaction is addressed by a technical protocol.

- Ecore Type: EEnum

ProtocolType			
Item			
Name	Literal	Value	Description
Other	Other	0	The type of protocol is unknown or not covered by any of the other types listed
Transport	Transport	1	The protocol addresses transport-level aspects of communication with the service; ***example:*** *TCP, HTTP*
Message	Message	2	The protocol addresses message-level aspects of communication with the service; ***example:*** *SOAP, MTOM, WS-Addressing*
Security	Security	3	The protocol addresses security aspects of communication with the service; ***example:*** *WS-SecureConversation, WS-Trust, SSL, TLS*
MetadataExchange	Metadata Exchange	4	The protocol addresses metadata exchange aspects of interaction with the service; ***example:*** *WS-Policy, WS-MetadataExchange*
ReliableMessaging	Reliable Messaging	5	The protocol addresses reliable messaging aspects of communication with the service; ***example:*** *WS-ReliableMessaging*
Transaction	Transaction	6	The protocol addresses transaction aspects of interaction with the service; ***example:*** *WS-Coordination, WS-AtomicTransation, WS-BusinessActivity*
Management	Management	7	The protocol addresses management aspects of interaction with the service; ***example:*** *WS-Management, WS-ResourceTransfer*

3 References

(Oaks et al., 2003) P. Oaks, A.H.M. Hofstede, D. Edmond, "Capabilities: describing what services can do", Proceedings of Service-Oriented Computing – ICSOC 2003, LNCS 2910, Springer, 2003, pp. 1-16.