Unified Service Description Language (USDL) Functional Module

February 28, 2011
Last Changed: May 24, 2011

Abstract. This document describes 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; an approach, which considers and connects business, operational (functional) and technical aspects of service description. The Functional Module allows capturing the business functions and capabilities of a service.

Table of Contents

Acknowledgements

The information in this document details a publicly released specification of the Unified Service Description Language (USDL) for dissemination and further exploitation through the Internet of Services Community in accordance with the terms set forth below. This document does not represent any prior commitment for standardization or implementation of any portion of this specification by SAP.

The information in this document was developed through the following publicly co-funded research projects THESEUS/TEXO, Australian Smart Services CRC, Premium Services, SLA@SOI.

THESEUS/TEXO is research project funded by the German Federal Ministry for Economics and Technology.

Premium Services is research project funded by the German Federal Ministry for Education and Research.

Australian Smart Services CRC is research and development partnership funded by the private sector and governments under the Australian Government's Cooperative Research Centre program.

SLA@SOI is a research project funded by the European Commission under the 7th Framework Programme.

The contributing authors are: Alistair Barros (SAP), Christian Baumann (SAP), Anis Charfi (SAP), Steffen Heinzl (SAP), Tom Kiemes (SAP), Uwe Kylau (SAP), Norman May (SAP), Oliver Müller (SAP, ERCIS Münster[1]), Francesco Novelli (SAP), Daniel Oberle (SAP), Philip Robinson (SAP), Benjamin Schmeling (SAP), Wolfgang Theilmann (SAP), Heiko Witteborg (SAP).

[1] European Research Center for Information Systems at the Westfälische Wilhelms-Universität Münster

Terms of Use Agreement

IMPORTANT- PLEASE READ CAREFULLY: THIS TERMS OF USE AGREEMENT ("AGREEMENT") IS A BINDING AGREEMENT BETWEEN SAP AG, A GERMAN COMPANY WITH OFFICES AT DIETMAR-HOPP-ALLEE 16, 69190 WALLDORF, GERMANY ("SAP"), AND YOU, BEING EITHER AN INDIVIDUAL OR SINGLE LEGAL ENTITY ("YOU" OR "YOUR") REGARDING YOUR USE OF THIS DOCUMENT AND ANY INFORMATION CONTAINED THEREIN ("MATERIAL"):

BY DOWNLOADING, COPYING, OR OTHERWISE USING THE MATERIAL, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT DOWNLOAD, COPY, OR USE THE MATERIAL.

IF YOU ARE DOWNLOADING, COPYING, OR USING THE MATERIAL ON BEHALF OF YOUR EMPLOYER OR A S A CONSULTANT OR AGENT OF A THIRD PARTY (COLLECTIVELY "YOUR COMPANY"), YOU REPRESENT AND WARRANT THAT YOU HAVE THE AUTHORITY TO ACT ON BEHALF OF AND BIND YOUR COMPANY TO THE TERMS OF THIS AGREEMENT AND ANY REFERENCE TO YOU OR YOUR SHALL ALSO INCLUDE YOUR COMPANY.

1. "SAP ENTITY/ENTITIES" shall mean SAP's affiliates and its subsidiaries, defined as corporations or other entities of which SAP owns, either directly or indirectly, more than fifty percent (50%) of the stock or other equity interests.

   "SAP SOFTWARE" shall mean the software products of SAP and/or SAP ENTITIES marketed and licensed by SAP and/or SAP ENTITIES.

   "INTELLECTUAL PROPERTY RIGHTS" means patents of any type, design rights, utility models or other similar invention rights, copyrights, trademarks, service marks, trade secret or confidentiality rights, and any other intangible property rights including applications for any of the foregoing, in any country, arising under statutory or common law or by contract and whether or not perfected, now existing or hereafter filed, issued, or acquired.

2. SAP grants YOU a nonexclusive, royalty-free, fully paid up, worldwide right to use, copy, display, perform, transmit, translate and distribute the MATERIAL provided hereunder. This shall include the right to reproduce, adapt, modify and to create derivative works of the MATERIAL and to make, have made, offer to sell, sell, lease, or otherwise distribute any product, and to practice any method, embodying such MATERIAL (including the right to sublicense any of the foregoing rights).

   SAP reserves the right to modify, change or discontinue the MATERIAL without notice at any time.

   YOU must not remove, overprint or deface any notice of copyright, trademark, logo, legend, or other notice of ownership from any originals or copies of the MATERIAL accessed hereunder. YOU agree to comply with the terms of the SAP Copyright Policy and those terms found by clicking on Copyright/Trademark at the web page http://www.internet-of-services.com.

   FOR AVOIDANCE OF DOUBT, NOTHING IN THIS AGREEMENT SHALL BE DEEMED TO

   (I) TO ASSUME OR PROVIDE FOR THE TRANSFER OF OWNERSHIP OF ANY INTELLECTUAL PROPERTY RIGHTS. ALL INTELLECTUAL PROPERTY RIGHTS INCLUDED, WITHOUT LIMITATION, COPYRIGHT IN ANY MATERIAL PROVIDED HEREUNDER, SHALL VEST IN AND AT ALL TIMES REMAIN VESTED IN THE ORIGINATOR OF THAT INTELLECTUAL PROPERTY RIGHT.

   (II) GIVE YOU THE RIGHT TO MODIFY, COPY, DISTRIBUTE, TRANSMIT, DISPLAY, PERFORM, REPRODUCE, PUBLISH, LICENSE, CREATE DERIVATIVE WORKS FROM,

TRANSFER; OR SELL ANY SAP SOFTWARE OR OTHER PRODUCT FOR ANY REASON UNLESS OTHERWISE PERMITTED BY SAP.

3. Any MATERIAL made available hereunder is provided to YOU "as is". SAP does not guarantee or warrant any features or qualities of the MATERIAL or give any undertaking with regard to any other quality. No warranty or undertaking shall be implied by YOU from any published MATERIAL except to the extent SAP has expressly confirmed such warranty or undertaking in writing. Warranties are validly given only with the express written confirmation of SAPs management.

   SAP does not represent or endorse the accuracy or reliability of any MATERIAL provided hereunder. SAP shall not be liable for damages caused by the use of the MATERIAL, unless such damages have been caused by SAPs willful misconduct.

   TO THE EXTENT ALLOWABLE BY APPLICABLE LAW, SAP AND ITS AFFILIATES, SUBSIDIARIES, OFFICERS, EMPLOYEES, AGENTS, PARTNERS, AND LICENSORS ARE NOT LIABLE TO YOU FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, CONSEQUENTIAL, OR EXEMPLARY DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS, REVENUE, GOODWILL, USE, DATA, OR OTHER INTANGIBLE LOSSES (EVEN IF SAP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES), HOWEVER CAUSED, WHETHER IN CONTRACT, TORT, OR OTHERWISE, ARISING OUT OF OR RESULTING FROM: (i) THE USE OR THE INABILITY TO USE THE MATERIAL; (ii) THE COST OF PROCUREMENT OF SUBSTITUTE GOODS AND SERVICES ARISING OUT OF YOUR USE OR INABILITY TO USE THE MATERIAL; OR (iii) ANY OTHER MATTER RELATING TO THE MATERIAL PROVIDED HEREUNDER. NOTWITHSTANDING ANYTHING TO THE CONTRARY HEREIN, THESE LIMITATIONS SHALL NOT APPLY IN CASE OF INTENT BY SAP AND IN CASE OF SAPS STATUTORY LIABILITY FOR PERSONAL INJURY AND DEFECTIVE PRODUCTS.

4. This AGREEMENT represents the complete and full agreement. No verbal side-agreements exist. Any changes to this AGREEMENT must be made in writing. This applies also to the revocation of the requirements for the written form.
5. This AGREEMENT shall be governed by the laws of Germany. The sole place of jurisdiction for all disputes arising directly in connection with this AGREEMENT shall be Karlsruhe, Germany.

1 Introduction

As outlined in the central document of this series *"USDL Overview"*, 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. Their purpose is to provide 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 Overview"*.

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 9 modules in the set that constitutes USDL version 3.0.

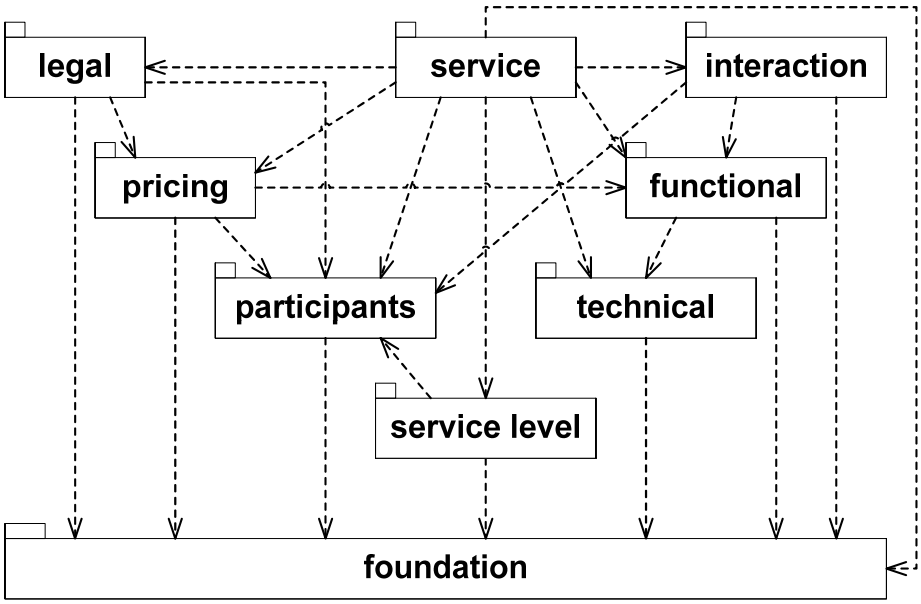

Figure 1 Packages comprising the USDL model and their dependencies (represented as arrows)

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 Functional Module defined in package "functional". The series also includes:

- *USDL Overview*
- *Module-specific documentation of the modules Foundation, Service, Participants, Technical, Interaction, Pricing, Service Level , and Legal*

The document only provides insights into the concepts of the Functional Module. For a complete overview of USDL it is recommended to also consider the other documents of the series.

2 Overview

2.1 Introduction to Functional Module

One of the most integral parts of every service description is to express *what* it is that a service will achieve for the beneficiaries involved (e.g. customers), i.e. its value proposition. In order to equally enable the description of human and automated services, the Functional Module captures such service functionality in a conceptual way. Conceptual in this context means independent of the ways to technically access functionality (the *how* part). 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, via 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.

Capability modeling usually takes a *black-box* view because it only captures the capability as something that is externally visible and does not reveal how it is realized internally. Other approaches aimed at modeling functionality, e.g. software function/component models and business process models, provide a more detailed formalization. These approaches trace their roots to structured systems analysis and design techniques, which introduced the principle of functional decomposition, and offer a *white-box* or *gray-box* view depending on the level of detail provided about the components/processes. For example, drilling down to the lowest level of atomic functions performed in an organization, including information about the roles and systems involved, can be regarded as a true *white-box* (sometimes also called *glass-box*) view. However, independent of how much is revealed about internal structures, there is always a set of high-level functions at the top which are offered to external parties as the capabilities of the organization.

It should be pointed out that capability modeling does not have to be limited to a flat, single-layer model, as well. In fact, there are approaches that propose quite diverse hierarchical models, which also capture interconnections between individual capabilities in terms of inputs, outputs and exceptions. In using the principles of decomposition, they slightly extend their scope from *black-box* to *gray-box* and thus share similarities with function or component modeling. The difference is that they describe functionality from a business point of view, which does not go beyond a certain level of detail, as opposed to an IT systems point of view that usually covers all aspects of realization/implementation.

For USDL a mix of capability modeling and function modeling has been chosen. In particular, service functionality is modeled as a set of hierarchical functions, which, at the top-most level, are externally visible as capabilities. The reason for this conceptualization is that while most service consumers might not care how functions offered as capabilities are structured internally, such information is of

interest to intermediaries aggregating, re-purposing and enriching a service. For example, a service broker that provides payment and billing facilities supports fine-grained payment models, e.g. collecting multiple apportions during service execution. In order for it to integrate such a payment model correctly, it requires detailed knowledge about the structure of a capability.

As outlined previously, functions are the building blocks of rendering a capability and have a number of inherent characteristics, some of which are similar to concepts of technical interfaces. Functions produce outcome, e.g., something is created, transformed, delivered or destroyed. Functions are performed by some actor (agent), who/which in doing so usually operates on one or more objects (resources), consuming and producing some of the objects, while others are only affected. It is furthermore common that actors use resources as tools to perform an action. In some cases it is even 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.

2.2 General Module Information

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. A class diagram of the package is depicted in Figure 2. The diagram shows which associations are compositions and which ones are normal relationships. Associations not shown are assumed to be of type composition by default.

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.

2.3 Module Dependencies

In order to understand concepts from referenced USDL modules in detail, it is recommended to read the following documents, which cover other USDL modules:

- Foundation
- Service
- Technical

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

Name	Type	Module	Description
NetworkProvisionedEntity	Abstract EClass	Service	The central concept of the USDL model that represents all entities provisioned into a service network, e.g. service or service bundle
Service	EClass	Service	A network-provisioned entity that offers capabilities, which are exposed through a technical interface
Description	EClass	Foundation	A generic concept that provides various information elements to describe USDL objects
Resource	EClass	Foundation	A generic concept to represent classes of concrete 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
Artifact	EClass	Foundation	A generic concept that allows to point to service metadata outside of USDL, as well as arbitrary documents, files, web pages, etc.
Option	Abstract EClass	Foundation	A concept to define subsets of service features and characteristics, in order to create variants of a service and thus only offer parts of the service
Classification	EClass	Foundation	A generic concept that can be used to classify USDL objects into defined classification systems
TypeReference	EClass	Foundation	A specific type of classification that represents a class/type in a type system, e.g. XML schema instance
VariableDeclaration	EClass	Foundation	A concept to capture declaration of generic variables
Condition	EClass	Foundation	A generic concept to capture conditions, i.e. state of objects that is associated with contextual meaning
FunctionalElementRef	Interface EClass	Foundation	The super type of all USDL classes capturing abstract functionality that can be exposed through a technical interface or parts thereof
ServiceLevelElementRef	Interface EClass	Foundation	The super type of all USDL classes that represent concepts to which a service level attribute may apply
Interface	EClass	Technical	A concept to capture relevant details about how to technically access service functionality

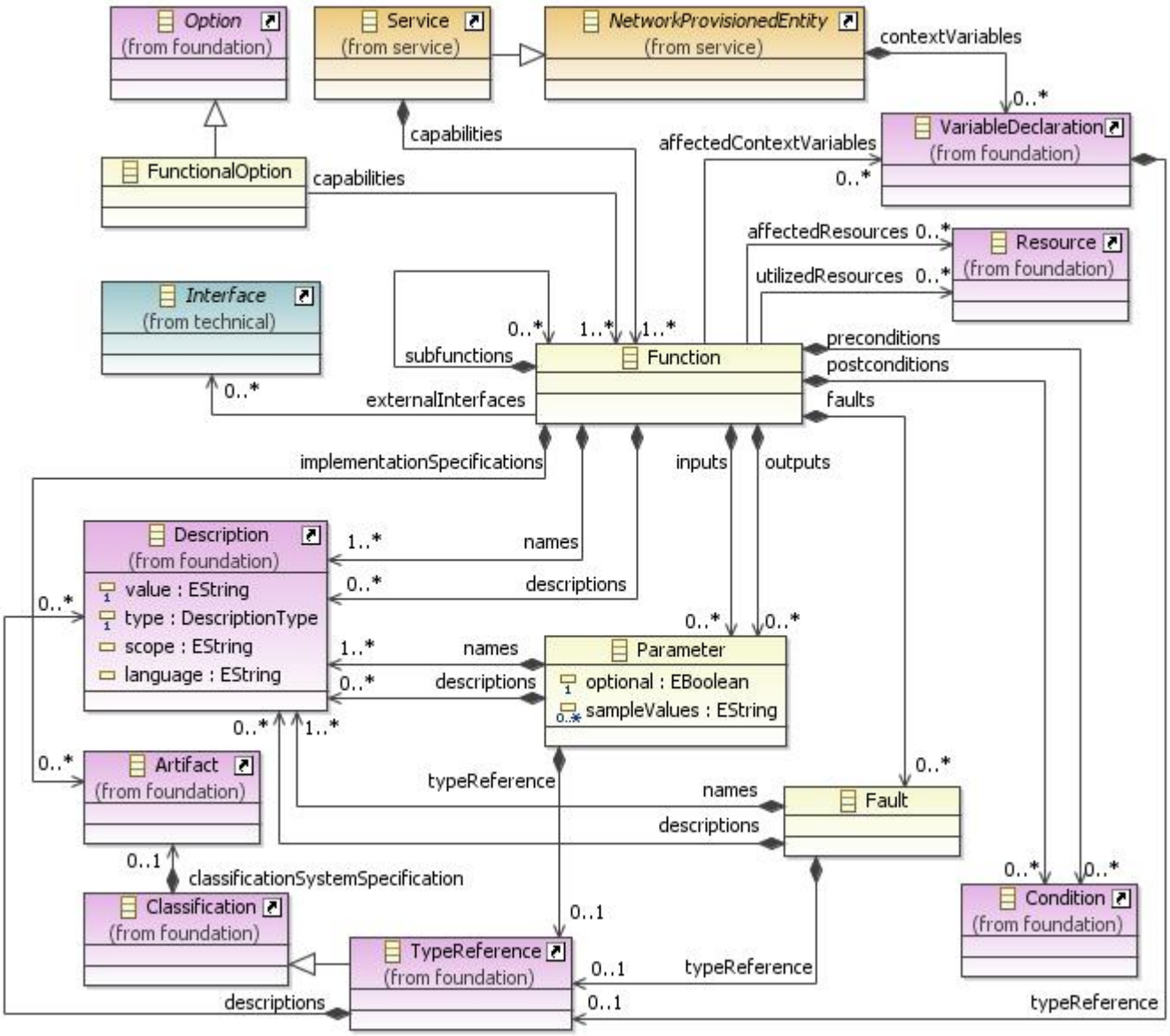

Figure 2 Class diagram of the package that captures the Functional Module

3 Functional Module: Model

3.1 Function

Function is used to capture an informal description of what the service does, i.e., its core functionality. A Function is an entity of activity that is performed by an actor (agent). Functions that are available to external parties, e.g. partners in a business network, are understood as capabilities. In this context a Function expresses 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.

Apart from this mandatory requirement, USDL offers much flexibility in describing the conceptual side of a service. Functions can be decomposed explicitly into lower-level function blocks (sub-functions), with the possibility to include descriptions of input/output parameters, faults and conditions on each layer. If such a *grey-box* view is not desired, a traditional *black-box* approach can be taken (only describing top-level functions, i.e. capabilities). Alternatively, USDL also offers a way to provide a *white-box* view using the artifact concept to include a link to an external, possibly complete, specification of the function's implementation.

Furthermore, functions may be associated with resource descriptions in order to capture information about the resource objects upon which actors operate during service execution (rendering of capabilities), e.g., which they transform or manipulate in terms of appearance, state, etc. This might also involve other resources, which are utilized during operation, e.g. as tools.

Example 1: Project Management

As part of project management the capability "create project time plan" is rendered. In a series of several steps (individual sub-functions) the detailed time plan of the project is produced (output), taking into account (input) parameters like project goals, activities / work items, available resources (workforce, budget, ...), and project duration.

Example 2: Banking

A bank offers the capability "open term deposit account" which allows service consumers to open such an account for 3, 6, or 12 months. The capability can be decomposed into several sub-functions such as "collect personal details" or the actual opening of the account. Regarding the first function, a required input is "personal details", for instance.

Note: Entities that reference function objects contain these objects, respectively complete function hierarchies. This ensures that top-level functions are known and interpreted correctly, i.e. as capabilities.

- Ecore Type: EClass
- Interfaces: FunctionalElementRef, ServiceLevelElementRef
- Superclass: N/A

Function			
Relations			
Name	Type	Cardinality	Description
names	Description	1..*	The set of names of the function; *constraint: type* of description has to be set to *name*
subfunctions	Function	0..*	The set of functional building blocks of the function
inputs	Parameter	0..*	The set of input parameters required for performing the function
outputs	Parameter	0..*	The set of output parameters produced by performing the function

preconditions	Condition	0..*	The set of conditions that have to be satisfied before the function can be performed
postconditions	Condition	0..*	The set of conditions that hold after the function is completed successfully
faults	Fault	0..*	The set of faults that may occur during performing the function
affectedContextVariables	Variable Declaration	0..*	The set of context variables that potentially change as part of performing the function
affectedResources	Resource	0..*	The set of resources transformed/manipulated as part of performing the function
utilizedResources	Resource	0..*	The set of resources that are utilized as part of performing the function
implementation Specifications	Artifact	0..*	Link to a set of formal specifications that define how the function is implemented
externalInterfaces	Interface	0..*	Reference to a set of separate (technical) interfaces that allow access to a function; *constraint:* function is a top-level function (capability)
descriptions	Description	0..*	Set of (additional) descriptive information about the function, possibly in multiple natural languages

Examples (in pseudo concrete syntax)

```
<identifiableElement xsi:type=”service:Service”>
...
 <contextVariables>
  <variableDeclaration xsi:id=”accountID”>
   <name>
    <value> accountID </value>
    <type> name </type>
   </name>
  </variableDeclaration>
  ...
 </contextVariables>

 <capabilities>
  <function xsi:id=”func345”>

   <names>
    <description>
     <value> Open Term Deposit Account </value>
     <type> name </type>
     <language> en </language>
    </description>
   </names>

   <externalInterfaces> intf8642 </externalInterfaces>

   <descriptions>
    <description>
     <value> Consumers are able to open a term deposit account for 3, 6, or 12 months. </value>
     <type> freetextLong </type>
     <language> en </language>
    </description>
   </descriptions>

   <affectedContextVariables> accountID, ... </affectedContextVariables>
```

```
    <subfunctions>

     <!-- sub-function #1 -->

     <function xsi:id="func433">
      <names>
       <description>
        <value> Collect Personal Details </value>
        <type> name </type>
        <language> en </language>
       </description>
      </names>
      <descriptions>
       <description>
        <value> The first step of application is to collect personal details from the customer. </value>
        <type> freetextLong </type>
        <language> en </language>
       </description>
      </descriptions>
      <outputs>
       <parameter> ... </parameter>
      </outputs>
     </function>
      ...

     <!-- sub-function #3 -->

     <function xsi:id="func435">
      <names>
       <description>
        <value> Open Account </value>
        <type> name </type>
        <language> en </language>
       </description>
      </names>
      <descriptions>
       <description>
        <value> With all details collected, the account can be created and provisioned. </value>
        <type> freetextLong </type>
        <language> en </language>
       </description>
      </descriptions>
      <inputs>
       <parameter> ... </parameter>
       <parameter> ... </parameter>
      </inputs>
      <outputs>
       <parameter> ... </parameter>
      </outputs>
     </function>

    </subfunctions>
   ...
   </function>
  </capabilities>
...
</identifiableElement>
```

3.2 Parameter

Parameter is used to capture conceptual input to and output of functions. Parameters, on the one hand, may 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: FunctionalElementRef, ServiceLevelElementRef, CopyrightProtectedElement
- Superclass: N/A

Parameter			
Attributes			
Name	Type	Cardinality	Description
optional	EBoolean	1	This flag indicates whether the parameter has to be present when the function is invoked or whether it can be omitted
sampleValues	EString	0..*	List of sample values given in an informal description
Relations			
Name	Type	Cardinality	Description
names	Description	1..*	The set of names of the parameter; *constraint: type* of description has to be set to *name*
typeReference	Type Reference	0..1	A pointer to a an entity in a type schema that formally specifies the structure of the parameter
descriptions	Description	0..*	Set of (additional) descriptive information about the parameter, possibly in multiple natural languages

Examples (in pseudo concrete syntax)

```
...
<function>
...
 <inputs>
  <parameter xsi:id="param876">
   <names>
    <description>
     <value> Personal Details </value>
     <type> name </type>
     <language> en </language>
    </description>
   </names>

   <typeReference>
    <classificationSystemID>
      http://www.moonbank.com/banking/businessObjects
    </classificationSystemID>
    <classID> Customer </classID>
   </typeReference>

   <descriptions>
    <description>
     <value> Personal details of a person </value>
     <type> freetextShort </type>
     <language> en </language>
```

```
      </description>
    </descriptions>
  </parameter>
...
 </inputs>
...
</function>
```

3.3 Fault

Fault is used to capture information about conceptual faults/exceptions that may occur when a function is performed.

- Ecore Type: EClass
- Interfaces: FunctionalElementRef, ServiceLevelElementRef
- Superclass: N/A

Fault			
Relations			
Name	Type	Cardinality	Description
names	Description	1..*	The set of names of the fault; *constraint: type* of description has to be set to *name*
typeReference	Type Reference	0..1	A pointer to a an entity in a type schema that formally specifies the structure of the fault
descriptions	Description	0..*	Set of (additional) descriptive information about the fault, possibly in multiple natural languages
Examples (in pseudo concrete syntax)			

3.4 FunctionalOption

FunctionalOption is a concrete service option that defines a subset from the overall functionality offered by a service, i.e. a subset of the service's capabilities.

Please refer to the USDL Foundation for further details about the concepts of service options.

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

FunctionalOption			
Relations			
Name	Type	Cardinality	Description
capabilities	Function	1..*	The subset of the service's capabilities (i.e. partial functionality), which are defined as an option; *constraint:* only top-level functions to be referenced
Examples (in pseudo concrete syntax)			