Unified Service Description Language (USDL) Core Module

December 28, 2009

Edited by SAP Research

Abstract. This document describes what is called the Core 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 Core module covers concepts central to USDL, most important service and service bundle. The module also includes some operational aspects of these concepts. Among them are dimensions of service typing, e.g. functional decomposition, and dependencies between services and other entities.

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.

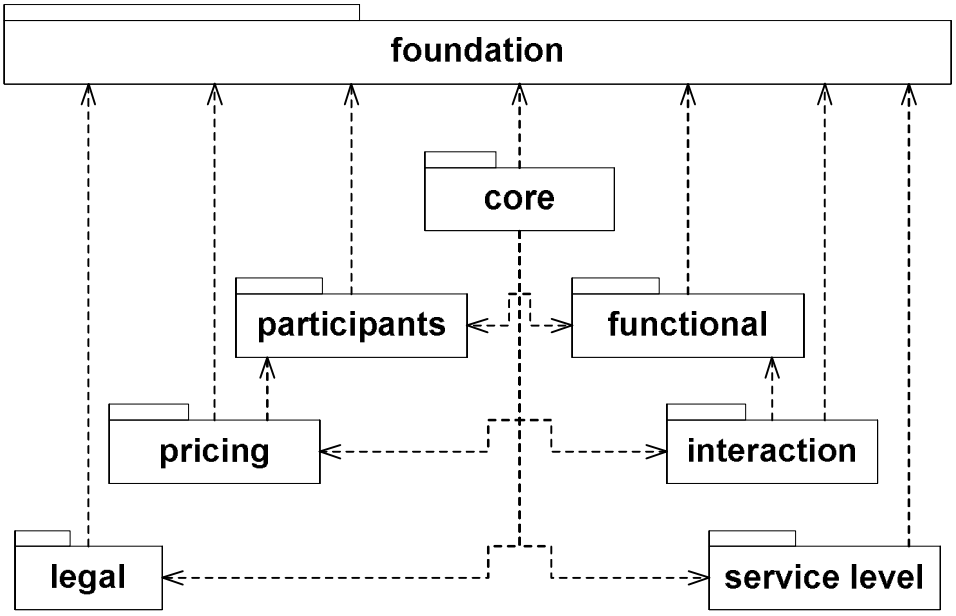

Figure 1 Packages comprising the USDL model and their dependencies (represented as arrows)

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*, *Participants*, *Functional*, *Interaction*, *Pricing*, *Service Level* (includes geographical and temporal availability) and *Legal*

The document only provides insights into the concepts of the Core 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 Core Module

The Core module can be regarded as the center of USDL. It ties together all the aspects of service description that are distributed across other USDL modules. At the heart of the Core module are concepts that represent the entities that USDL is actually describing, namely entities provisioned into service networks (e.g. services, service bundles). It is important to understand that in this context only network level aspects of service description are of interest. Agents active in a service network need to communicate and understand aspects including:

- Service Release: What is the service? Who is able to use a service for which purposes (consumption, aggregation, channeling, etc.)? What are the conditions for usage?
- Service Access: How, where and when is a service made available?
- Service Consumption: How is a service successfully consumed? What is where and when expected by different parties participating in service provisioning/delivery before, during and after the execution of the service?

Out of scope of USDL are aspects only relevant before the service is provisioned into the network, e.g. within the organization that provides the service. For example, it is neither considered how a service is developed, nor how its quality is assured internally.

Network provisioned entities are detailed along various dimensions of typing, i.e. general service nature, functional granularity and decomposition. Out of these three types, decomposition had the most profound impact onto the model. By default, a service is treated as an indivisible (atomic)

entity, meaning it is not subject to decomposition, even though its internal implementation may be arbitrary complex. A composite service, in turn, is a specialization of this concept, as it consists of other components that are known in the ecosystem of the application processing the service description. In order to allow for flexible composition, a component may take the form of a concrete service (atomic or composite), a service bundle or an abstract service. All three of them implement an interface (Composable) for this purpose. Composite services may impose order constraints upon their components in the form of a simple process definition.

The concept of abstract services is used to define templates that describe classes of services. The idea is to pre-set certain service aspects so that an abstract service may serve as a placeholder for concrete services. The placeholder can then be bound to a concrete service at some point in time (e.g. later than design time).

Another important design decision was to treat functionality-based and business-based decomposition as two separate concepts. This led to the division of tradable entities into service and service bundle. The main reason is that a composite service combines individual services to provide new capabilities, i.e. the component services work together to achieve a higher function. A service bundle on the other hand only combines services for commercial reasons, e.g. competitive pricing or increase of sales numbers. The services are functionally independent and do not influence each other in any way. In contrast, services aggregated in a composition usually have dependencies between each other and may even be aware that they are executed in the context of a composition.

The dimension of service nature was mentioned in the *"USDL Technical Overview Paper"*. It generally distinguishes automated, semi-automated and human services. Adding the aspect of granularity, i.e. functional complexity, to this, services can be characterized as either providing an elemental function or a more comprehensive set of capabilities. The latter may involve a resource that is managed or exposed by the service.

Common among all top-level service concepts (service, service bundle, abstract service) is that they may have dependencies with other entities. This does not always mean that one service depends on another service. In fact, quite a number of different dependencies can be observed. Services may require another service or a resource. One service may enhance another one, i.e. provide additional capabilities, or may be suitable to substitute another service although its functionality is quite different. The dependency concept captures the dependencies that services, service bundles or abstract services have.

2.1 Module Info

Parameters of the package that captures the module

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

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

- Participants
- Functional
- Interaction
- Pricing
- Service Level
- Legal

A quick overview of the concepts most widely used in the Core 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.
DependencyTarget	Interface EClass	Foundation	Serves as the super type of all elements of USDL that can be the target of a dependency
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
Classification	EClass	Foundation	A generic concept that can be used to classify description elements into defined taxonomies
Certification	EClass	Foundation	A generic concept that can be used to associate description elements with certifications they hold
OrderType	EEnum	Foundation	Indicates the type of ordered structures, which at the moment are confined to simple block-structured constructs (e.g. sequence, parallel flow, choice)

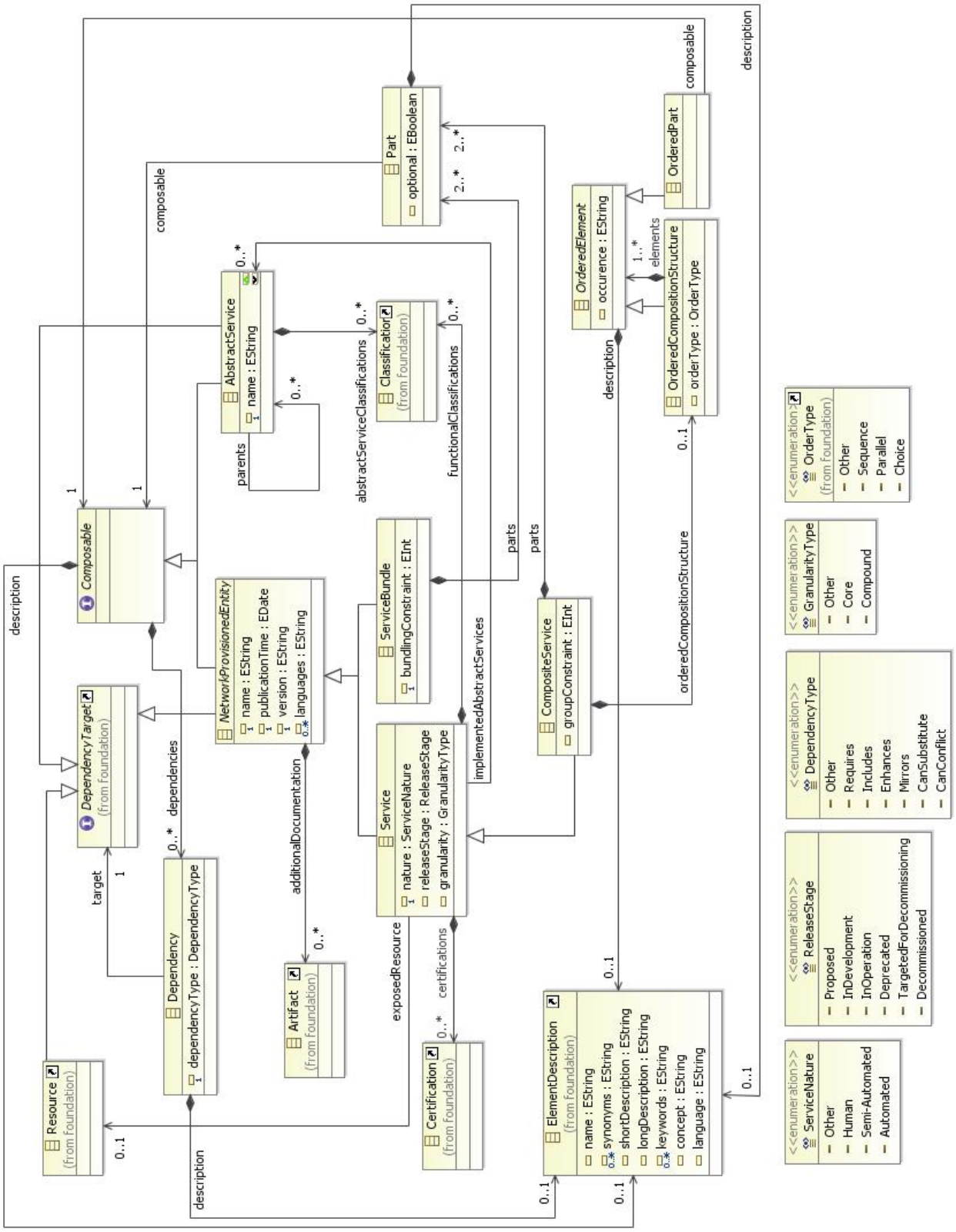

Figure 2 Class diagram of the package that captures the Core module

2.3 Composable

Composable serves as the super type of all USDL elements/concepts that can be included in a composite service or service bundle, namely **Service**, **ServiceBundle** and **AbstractService**.

For modeling simplicity, a **Composable** may reference an element description, which contains various information items that can be used to describe services, service bundles and abstract services. For the same reasons, a **Composable** may also specify dependencies on other elements of the following types: **Service**, **ServiceBundle**, **AbstractService** or **Resource**.

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

Composable			
Relations			
Name	Type	Cardinality	Description
dependencies	Dependency	0..*	The set of dependencies that services, service bundles and abstract services have onto other services, service bundles, abstract services and resources (Note: has been put here for convenience reasons)
description	Element Description	0..1	A description of the service, service bundle or abstract service
Examples (in pseudo concrete syntax)			

2.4 NetworkProvisionedEntity

NetworkProvisionedEntity is the central concept of the USDL meta model and represents all entities that are provisioned into, i.e. exposed to, a service network (e.g. service marketplace). The **NetworkProvisionedEntity** class serves as an abstract super type for all concrete exposable entities (**Service**, **Service Bundle**). It constitutes the main entry point into the model (core elements and elements captured in modules).

- Ecore Type: Abstract EClass
- Interfaces: IdentifiableElement, Composable, DependencyTarget
- Superclass: N/A

NetworkProvisionedEntity			
Attributes			
Name	Type	Cardinality	Description
name	EString	1	The name of the service or service bundle
version	EString	1	The functional version of the service or service bundle
publicationTime	EDate	1	The time when the service or service bundle (in a specific version) was made first available to the service network
languages	EString	0..*	A list of languages the service or service bundle is available in (ISO 639-1)

Relations			
Name	Type	Cardinality	Description
additionalDocumentation	Artifact	0..*	The set of optional links to additional information material that gives further description of the service, service demonstrations, reviews, etc.
requestTime	Time	0..*	TimeInstant, TimeInterval, or RecurrencePattern describing when the network provisioned entity may be requested
deliveryTime	Time	0..*	TimeInstant, TimeInterval, or RecurrencePattern describing when the network provisioned entity can be delivered
requestLocation	Location	0..*	Location from where the network provisioned entity may be requested
deliveryLocation	Location	0..*	Location where the network provisioned entity can be delivered to
pricePlans	PricePlan	0..*	The optional set of price plans that describe the price structure of the service or service bundle (see Pricing module for details)

Examples (in pseudo concrete syntax)

```
…
 <guid> P05529-RSM-DL006 </guid>
 <namespace> http://www.moonbank.com/BankingServicesNetwork/ <namespace>
 <name> Term Deposit </name>
 <version> 2.3.11 </version>
 <publicationTime> 2009-11-20T01:37:26.000-0600 </publicationTime>
 <languages> ENG </languages>

 <description>
  <synonyms> RB-COMP-Term_Deposit </synonyms>
  <synonyms> P05529-RSM-DL006 </synonyms>
  <shortDescription>
   Term Deposit service provides single-point management of term deposit accounts in retail banking.
  </shortDescription>
  <keywords> Term Deposit </keywords>
  <keywords> Retail </keywords>
  <keywords> Banking </keywords>
  <keywords> Account Management </keywords>
  <language> ENG </language>
 </description>

 <additionalDocumentation>
  <shortDescription></shortDescription>
  <type> TermsOfUse </type>
  <source> service provider </source>
  <mimeType> application/pdf </mimeType>
  <uri> http://www.moonbank.com/services/terms-of-use/general.pdf </uri>
  <description>
   <name> Moonbank Services General Terms of Use </name>
  </description>
 </additionalDocumentation>
```

(continues on next page)

```
<requestTime xsi:type=”foundation:DurationInterval”>
  <start xsi:type=”foundation:AbsolutePointInTime”>
    <value> 2010-01-01T02:00:00.000-0600 </value>
    <description>
      <name> Roll-out time </name>
    </description>
  </start>
  <intervalDuration>
    <type>year</type>
    <value>3.0</value>
  </intervalDuration>
  <description>
    <name> Service Validity Period </name>
    <shortDescription> The service will be available from 1st of January, 2010, 2:00am EST </shortDescription>
  </description>
</requestTime>

<deliveryLocation xsi:type=”foundation:AdministrativeArea”>
  <name> United States of America </name>
  <type> country </type>
  <description>
    <shortDescription>
      The service can only be delivered to applicants with a valid address in the USA.
    </shortDescription>
  </description>
</deliveryLocation>
…
```

2.5 Service

A **Service** is one type of entity provisioned into service networks (i.e. a sub type of **NetworkProvisionedEntity**). It exposes a set of capabilities that can be accessed through the service's own (abstract or technical) interface.

From the viewpoint of the service network (or partner ecosystem), it is either fundamental (atomic) or composed out of other entities (see **CompositeService**). Even in the case of a fundamental service, the internal implementation can be arbitrarily complex and the interface exposed by the service can entail structure.

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

Service			
Attributes			
Name	Type	Cardinality	Description
nature	ServiceNature	1	Nature indicates the general nature of the service (human, semi-automated, automated)
releaseStage	Release Stage	0..1	The service's stage in its development lifecycle
granularity	Granularity Type	0..1	Granularity indicates the functional granularity of the service

Relations			
Name	Type	Cardinality	Description
capabilities	Capability	1..*	The set of capabilities that constitute the service (i.e. the functionality offered by the service)
technicalInterfaces	Technical Interface	0..*	The set of technical interfaces exposed by the service; applies only to automated services
interactionProtocols	Interaction Protocol	0..*	The set of interaction protocols enacted by the service, which have to be followed to ensure successful use of the service's capabilities
functionalClassifications	Classification	0..*	The set of classifications into taxonomies the service is associated with
certifications	Certification	0..*	The set of certifications obtained for the service
implemented AbstractServices	Abstract Service	0..*	The set of abstract services (i.e. service templates) implemented/realized by the service
exposedResource	Resource	0..1	Optional reference to a resource, which is managed by the service (see also GranularityType *Compound*)

Examples (in pseudo concrete syntax)

```
<service>
…
 <nature> Semi-Automated </nature>
 <releaseStage> InOperation </releaseStage>
 <granularity> Compound </granularity>

 <functionalClassifications>
  <taxonomy> http://www.moonbank.com/BankingServicesNetwork/FunctionArea </taxonomy>
  <class> AccountManagement </class>
 </functionalClassifications>

 <certifications>
  <functionalClassifications>
   <taxonomy> http://www.fixprotocol.org/compliance/ </taxonomy>
   <class> Full </class>
  </functionalClassifications>
  <certificate>
   <type> Certificate </type>
   <mimeType> application/pdf </mimeType>
   <uri> http://www.moonbank.com/services/certificates/fix-full.pdf </uri>
   <description>
    <name> Moonbank Services – Full FIX Compliance Certificate </name>
   </description>
  </certificate>
 </certifications>

 <exposedResource>
  <name> Term Deposit Account </name>
  <type> SoftwareResource </type>
  <description>
   <shortDescription> A specialized account business object </shortDescription>
  </description>
 </exposedResource>
…
</service>
```

2.6 ServiceBundle

A **ServiceBundle** is a type of **NetworkProvisionedEntity** that contains other services, service bundles or abstract services available in a service network for delivery/consumption. Bundling is motivated by economic reasons, e.g. competitive pricing or combining a low-selling service with a highly demanded service in order to drive up sales figures of the former. The degree of collective delivery/consumption can be limited through a bundling constraint (all/some/one component(s) of the group).

Bundling abstract services allows for late binding of concrete services into the context of a bundle.

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

ServiceBundle			
Attributes			
Name	Type	Cardinality	Description
bundlingConstraint	EInt	1	A bundling constraint determines how many components of the bundle have to be selected for delivery. Having a constraint like this allows creating flexible bundling structures by nesting bundles. **valid values:** *-1*: all components need to be included *0*: any number of components can be included *1 and greater:* the number of components that needs to be selected
Relations			
Name	Type	Cardinality	Description
parts	Part	2..*	The set of parts that are bundled
Examples (in pseudo concrete syntax)			

2.7 CompositeService

A **CompositeService** is a nested service that aggregates other service, service bundles or abstract services. Unlike service bundles, which group services purely for business reasons, composite services provide new functionality by combining the capabilities of their components. The composition may be loose, i.e. does not follow a particular order, or it may be governed at its highest level by a process-based block-structured (hierarchical) order (*control flow*). Components may also influence each other by changing parts of the overall context the composition is executed in (*context dependencies*).

Composing abstract services that define the functionality (capabilities) composed allows for late binding of concrete services that realize this functionality.

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

CompositeService			
Attributes			
Name	Type	Cardinality	Description
groupingConstraint	EInt	1	A grouping constraint determines how many components of the composite service have to be selected for delivery. **valid values:** *-1*: all components need to be included *0*: any number of components can be included *1 and greater:* the number of components that needs to be selected
Relations			
Name	Type	Cardinality	Description
parts	Part	2..*	The set of parts that are composed into a composite
contextDependencies	Context Dependency	0..*	The set of direct context/data dependencies, which exist between composed services, bundles and/or abstract services
orderedCompositionStructure	Ordered Composition Structure	0..1	The top-most ordering structure, if the composite follow a defined control flow
Examples (in pseudo concrete syntax)			

2.8 AbstractService

AbstractService is used to represent classes of services, i.e. groups of services that comply with a number of predefined attributes. A configuration of attributes is provided via the same mechanism that is used to describe services, i.e. the class holds a set of references to other USDL elements, which are interpreted as templates. Services implementing the abstract service incorporate these templates into their description. However, not all elements can be predefined. Hence, AbstractService only comprises a subset of the attributes and relations accumulated at the concrete service level (class Service).

An abstract service may also be derived from other abstract services, whereby it inherits all definitions of its parents.

It should be noted that the attributes and relations inherited from the interfaces are to be interpreted as elements describing the abstract service, not the services implementing the abstract service. To be specific, an abstract service has a name, a unique ID and a description. It may also have dependencies to other services or abstract services.

- Ecore Type: EClass
- Interfaces: IdentifiableElement, Composable, DependencyTarget
- Superclass: N/A

AbstractService			
Attributes			
Name	Type	Cardinality	Description
nature	ServiceNature	0..1	Nature indicates the general nature of services implementing the abstract service
languages	EString	0..*	A list of natural languages that services implementing the abstract service need to be available in (ISO 639-1)

Relations			
Name	Type	Cardinality	Description
abstractService Classifications	Classification	0..*	The set of classifications into taxonomies of abstract service that the abstract service is associated with
parents	Abstract Service	0..*	The set of abstract services from which template descriptions are inherited
requestTime	Time	0..*	TimeInstant, TimeInterval, or RecurrencePattern describing when services implementing the abstract service may be requested
deliveryTime	Time	0..*	TimeInstant, TimeInterval, or RecurrencePattern describing when services implementing the abstract service can be delivered
requestLocation	Location	0..*	Location from where services implementing the abstract service may be requested
deliveryLocation	Location	0..*	Location where services implementing the abstract service can be delivered to
targetConsumers	Target Consumer	0..*	The set of user groups at which the services implementing the abstract service need to be targeted (see Participants module for details)
pricePlans	PricePlan	0..*	The set of price plans that describe the price structure of the services that implement the abstract service (see Pricing module for details)
capabilities	Capability	0..*	The set of capabilities that need to be provided by services implementing the abstract service
technicalInterfaces	Technical Interface	0..*	The set of technical interfaces exposed by services implementing the abstract service; applies only to automated services
interactionProtocols	Interaction Protocol	0..*	The set of interaction protocols enacted by services implementing the abstract service
functionalClassifications	Classification	0..*	The set of technical interfaces exposed by services implementing the abstract service; applies only to automated services
certifications	Certification	0..*	The set of certifications that need to be held by services implementing the abstract service
exposedResource	Resource	0..1	Reference to a resource, which is managed by services implementing the abstract service (see also GranularityType Compound)
Examples (in pseudo concrete syntax)			

2.9 Dependency

Dependency captures information about the relationships that exist between two elements of certain types (**Service**, **ServiceBundle**, **AbstractService** and **Resource**). This concept supports various types of dependencies such as requires, includes and mirrors, amongst others.

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

Dependency			
Attributes			
Name	Type	Cardinality	Description
dependencyType	Dependency Type	1	The type of dependency
Relations			
Name	Type	Cardinality	Description
target	Dependency Target	1	Reference to the target of the dependency
description	Element Description	0..1	A description of the dependency
Examples (in pseudo concrete syntax)			

2.10 Part

Part represents the concept of a slot within a composed structure. Using this abstraction layer on top of the actual service, service bundle or abstract service (that is part of the composed structure) allows for fine-grained configuration of a composition, e.g. mark a Part to be optional. It should be pointed out that general composition constraints (of a **CompositeService** or **ServiceBundle**) always take precedence over Part configurations. E.g., if the composition constraint is set to *all*, it is not possible to leave out optional Parts. In general, it is envisioned that Part configurations refine overall composition constraints and are not defined in a conflicting way.

Example: In order to express that a bundle offer comprises 5 out 7 different elements, of which 3 are mandatory, one would set bundlingConstraint to 5, mark 3 Parts mandatory and the other 4 Parts optional.

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

Part			
Attributes			
Name	Type	Cardinality	Description
optional	EBoolean	0..1	This flag indicates whether the component is optional (*true*), i.e. may be left aside during delivery of the composite structure, or whether it is mandatory (*false*); not setting this attribute means that no assumptions should be made about whether the component is mandatory or optional
Relations			
Name	Type	Cardinality	Description
composable	Composable	1	The reference to the service, service bundle or abstract service that fills this slot in the composition structure
description	Element Description	0..1	A description of the part
Examples (in pseudo concrete syntax)			

2.11 OrderedElement

OrderedElement serves as the super type of all USDL elements/concepts that can be part of an ordered structure of parts.

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

OrderedElement			
Attributes			
Name	Type	Cardinality	Description
occurrence	EInt	1	The number of occurrences of the element; can be used to express finite loops **valid values:** positive, non-null integers
Relations			
Name	Type	Cardinality	Description
description	Element Description	0..1	A description of the ordered element
Examples (in pseudo concrete syntax)			

2.12 OrderedCompositionStructure

OrderedCompositionStructure represents a composition of **OrderedElements** in a specific block-structured order. Examples of order types include sequential, parallel and choice. An **OrderedCompositionStructure** may be used by a composite service.

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

OrderedCompositionStructure			
Attributes			
Name	Type	Cardinality	Description
orderType	OrderType	1	Order type determines the type of ordered composition structure
Relations			
Name	Type	Cardinality	Description
elements	Ordered Element	1..*	The parts of the ordered composition structure (lower-level structures or atomic parts, i.e. composables)
Examples (in pseudo concrete syntax)			

2.13 OrderedPart

OrderedPart represents the lowest level building block of an **OrderedCompositionStructure**. It captures a single "lower-level" service (or service operation) from which a composite service is assembled.

- Ecore Type: EClass

- Interfaces: N/A
- Superclass: OrderedElement

OrderedPart			
Relations			
Name	Type	Cardinality	Description
composable	Composable	1	The reference to the service, service bundle or abstract service that is delivered (fully or in part) at this given point in the ordered structure
Examples (in pseudo concrete syntax)			

2.14 ServiceNature

ServiceNature indicates the general nature of a service. The nature of a service refers to the manner in which the service is performed (i.e. completely manually (human), fully automated or a combination of both).

- Ecore Type: EEnum

ServiceNature			
Items			
Name	Literal	Value	Description
Other	Other	0	Unknown service nature.
Human	Human	1	Human services only require the intervention of human actors and represent people participating in a service delivery. Such services are purely manual and involve human decision making or informal communication.
Semi-automated	Semi-automated	2	Semi-automated services take place with the participation of human beings and machines or IT applications.
Automated	Automated	3	Automated services denote that no human intervention is involved. It may be, for example, either an invoked IT application or a machine.

2.15 ReleaseStage

ReleaseStage indicates the releasing stage in the service network. It is generally defined with respect to a service delivery lifecycle because services are versioned into different release stages for delivery purposes.

- Ecore Type: EEnum

ReleaseStage			
Items			
Name	Literal	Value	Description
Proposed	Proposed	0	The service is at the idea or conception stage, and not yet in development
InDevelopment	In Development	1	The service is under development
InOperation	In Operation	2	The service is in operation

Deprecated	Deprecated	3	The service has been deprecated by a new functional version or another service
TargetedFor Decommissioning	Targeted For Decommissioning	4	The service is still operational, but will be decommissioned in the near future
Decommissioned	Decommissioned	5	The service has been deactivated and was decommissioned from the ecosystem

2.16 DependencyType

DependencyType indicates the semantics of a dependency.

- Ecore Type: EEnum

DependencyType			
Items			
Name	Literal	Value	Description
Other	Other	0	Unknown dependency type
Requires	Requires	1	Operation of the service, service bundle or abstract service depends on the operation of the referenced service, service bundle or abstract service; consequence is that the referenced object has to be available by the consumer (i.e. has to be ordered separately)
Includes	Includes	2	Operation of the service, service bundle or abstract service entails the operation of the referenced service, service bundle or abstract service; the referenced object is part of the service, service bundle or abstract service and does not need to be procured by the consumer
Enhances	Enhances	3	The service, service bundle or abstract service provides additional capabilities (functionality) on top of the referenced service, service bundle or abstract service
Mirrors	Mirrors	4	The service, service bundle or abstract service provides the same capabilities (functionality) as the referenced service, service bundle or abstract service and therefore constitutes an alternative to the referenced object
CanSubstitute	Can Substitute	5	The service, service bundle or abstract service provides capabilities (functionality) that are different from those of the referenced service, service bundle or abstract service, but achieves the same goals and therefore constitutes an alternative to the referenced object
CanConflict	Can Conflict	6	Operation of the service, service bundle or abstract service may conflict with the operation of the referenced service, service bundle or abstract service

2.17 GranularityType

GranularityType indicates the functional scope of a service from an external point of view (e.g. from a consumer's perspective). It does not necessarily reflect the functional complexity of a service. For

example, a complex composite (assembled out of several lower-level functions) may appear externally as just a single, simple function.

- Ecore Type: EEnum

Enum Name			
Items			
Name	Literal	Value	Description
Other	Other	0	Unknown granularity type
Core	Core	1	The service provides a single function from an external perspective of the service. **examples:** - temperature at a certain location (in contrast to a full weather report) - account opening (in contrast to complete account management)
Compound	Compound	2	The service provides a range of functions from an external perspective of the service. For example, the range of functions may relate to one or many resources, information objects, logical set of functions, etc. **examples:** - management of customer master data (entire CRUD interface plus auxiliary functions) - project management