# Unified Service Description Language (USDL) Interaction Module

December 28, 2009

Edited by SAP Research

**Abstract.** This document describes what is called the Interaction 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 Interaction module covers concepts that describe how consumers should communicate with complex services with respect to individual interaction points. These interaction points usually occur in a certain order, which has to be followed to successfully complete the service.

## 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.

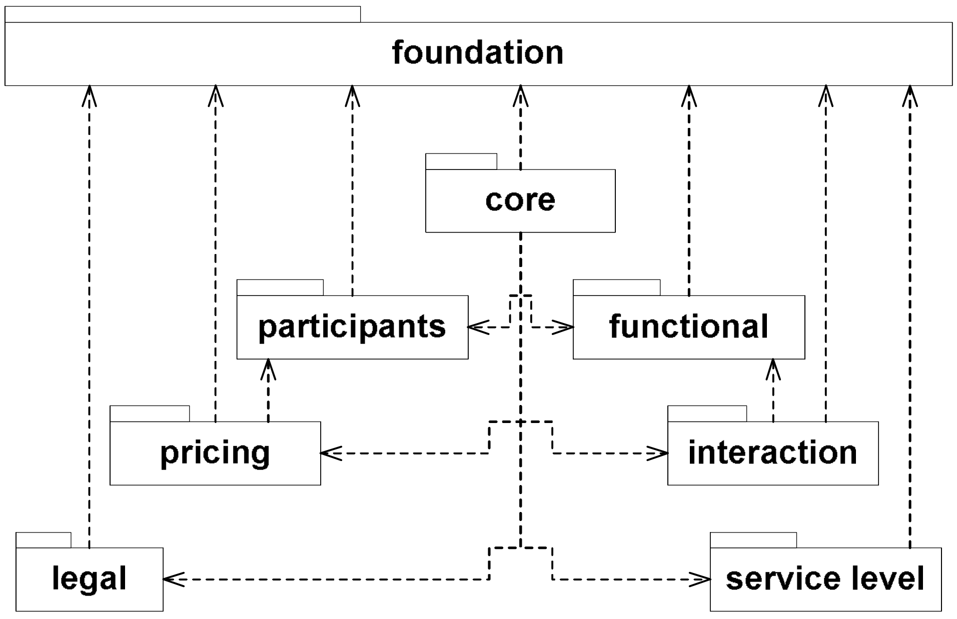

Figure 1 Packages comprising the USDL model and their dependencies (represented as arrows)

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

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

Knowledge about the capabilities provided by a service and the means of communication employed to access it, is one side of consuming a service. In addition, it might be necessary to also know in what order individual actions are performed and when it is necessary to interact with the service (respectively the actor performing the service), in order to trigger its execution to continue. These aspects are represented in an interaction protocol[1] and apply to services that comprise a complex course of action, at whose end one or more capabilities are successfully provided. In simple words, the interaction protocol captures the externally observable behavior of services.

The elemental building block of interaction protocols is the interaction. It is defined as an act of communication between the service consumer and one or more actors participating in the delivery of the service. This could be as simple as a consumer submitting a form to a front-office agency that channels the service. On the other hand, an interaction may involve a chain of intermediaries processing the message sent by the consumer before it reaches the service provider, in which case a core service action is performed. It should be noted that not every interaction necessarily triggers an action. However a series of interactions ultimately will have to result in a set of actions being performed, otherwise the service's capability will never be rendered.

An inherent characteristic of all interaction protocols is that they need to express an order among the interactions. To address this requirement we follow a flow-based approach as known from workflow and process languages (such as WS-BPEL). However, USDL does not intend to provide a

[1] Alternative terms are, for example, business protocol, conversation protocol, or public service view.

comprehensive formal specification of the interaction protocol but rather give an abstract high-level view. If there is a formal definition (e.g., an abstract process in WS-BPEL or an interaction protocol in some choreography language), a pointer to the artifact that contains it should be given. Formal definitions are likely to occur with automated services implemented in software, but are less likely in the case of human services. USDL targets both and thus provides the means to formally model simple interaction protocols. This should be sufficient to cover the needs of services that have no defined interaction protocol, but need to specify one in order to be properly provisioned into service networks.

As interaction protocols become easily very complex, especially for long-running composite services, they can be divided into self-contained higher-level sections. These are called *phases* in USDL and group interactions. A phase adds meaning to a low-granular portion of the interaction protocol by indicating the progress of service execution and by defining clear milestones, which are reached at the end of a phase.

## 2.1 Module Info

Parameters of the package that captures the module

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

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
- Functional
- Participants

A quick overview of the concepts most widely used in the Interaction 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. |
| Role | Abstract EClass | Participants | Serves as the super type of roles that participate in the provisioning/delivery of a network provisioned entity |

| | | | |
|---|---|---|---|
| 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) |
| Capability | EClass | Functional | The conceptualization of what the service does in terms of a course of action |
| Action | EClass | Functional | An elemental action performed to achieve a capability |

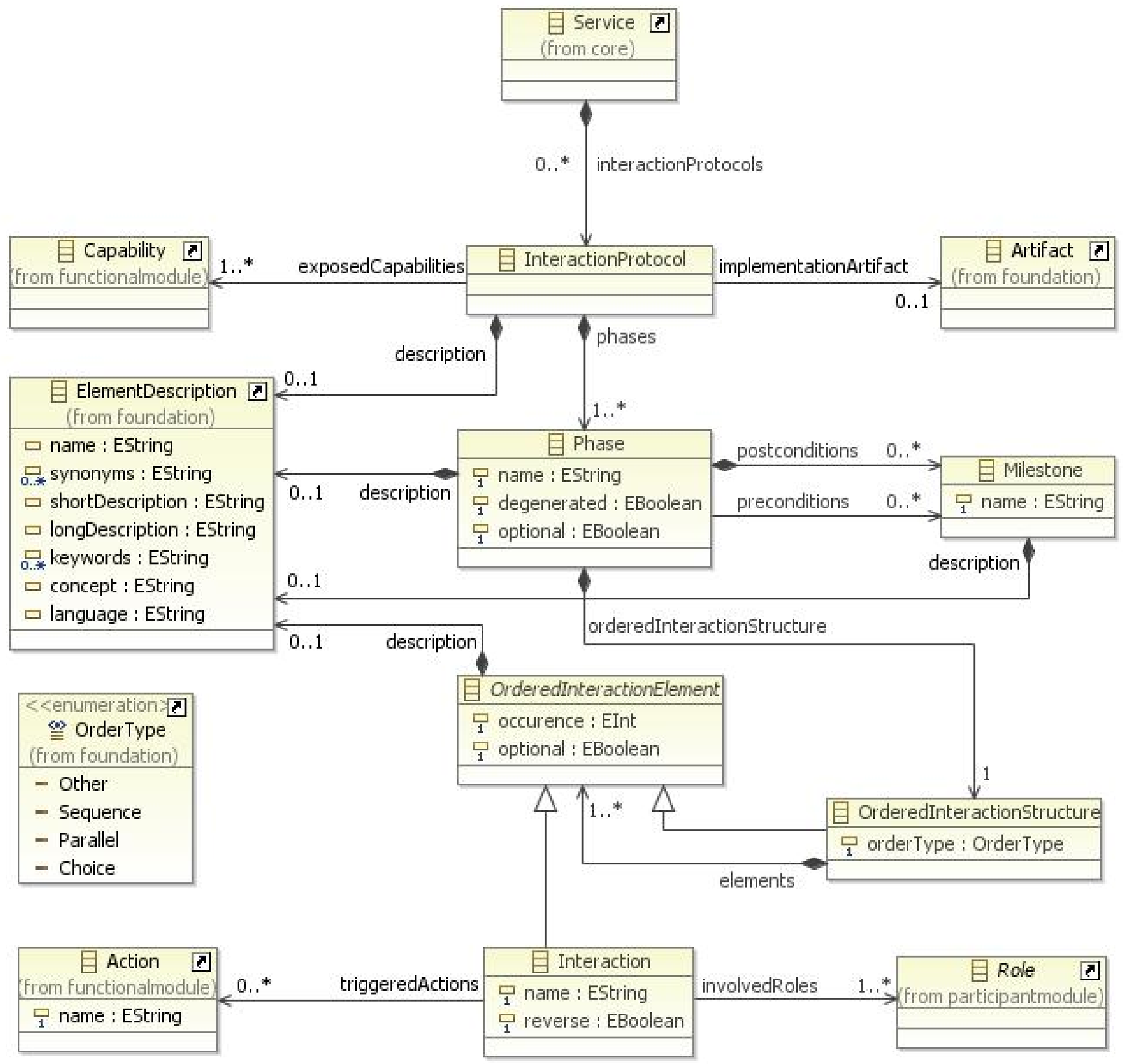

Figure 2 Class diagram of the package that captures the Interaction module

### 2.3 InteractionProtocol

**InteractionProtocol** represents a concept that captures the externally observable behavior of a service in terms of atomic communication acts - interactions. The concept is also known as interaction protocol, business protocol, public view or conversation protocol.

Of course, not every service requires a complex series of interactions in order to successfully provide its capabilities. Hence, **InteractionProtocol** is an optional element. Nonetheless, a service may choose to expose individual capabilities through dedicated interaction protocols, i.e. have more than one interaction protocol.

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

| InteractionProtocol | | | |
|---|---|---|---|
| Relations | | | |
| Name | Type | Cardinality | Description |
| exposedCapabilities | Capability | 1..* | The interaction protocol exposes at least one capability of the service, i.e. the successful completion of the protocol will result in the delivery of the referenced capabilities |
| phases | Phase | 1..* | The interaction protocol is divided into a sequentially ordered set of phases |
| implementationArtifact | Artifact | 0..1 | The interaction protocol may be formally specified in another formalism (e.g. BPEL) |
| description | Element Description | 0..1 | A description of the interaction protocol |
| Examples (in pseudo concrete syntax) | | | |

```
<service>
…
 <interactionProtocol>
  <exposedCapabilities> #cap345 </exposedCapabilities>
  <phases> … </phases>
  <phases> … </phases>
  <description>
   <shortDescription> Protocol to use when opening a new term deposit account </shortDescription>
  </description>
 </interactionProtocol>
…
</service>
```

### 2.4 Phase

A **Phase** groups a set of interactions, which together form a logical part of the overall interaction protocol associated with a meaning. The grouping is expressed with a block-structured (hierarchical) order.

Phases are useful to divide the interaction protocol of long-running and complex process-based services into smaller more comprehensible pieces. Usually separated by milestones, phases give service consumers a clear indication how far along in the execution of the service they are, what is happening, what is required from them and which service results are available at the end of the phase.

Obviously, interaction protocols that are simpler and only have a few interactions do not require phases. However, for reasons of model simplicity at least one phase has to be present. Thus, a phase can be marked as degenerate in order to indicate that it is just used as a container of interactions and does not possess any detailed semantics (apart from the fact that service capabilities are realized in the phase).

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

| Phase | | | |
|---|---|---|---|
| **Attributes** | | | |
| Name | Type | Cardinality | Description |
| name | EString | 1 | The name of the phase |
| degenerated | EBoolean | 1 | This attribute marks the phase as an actual or a degenerated phase |
| optional | EBoolean | 1 | Indicates whether the phase is optional, i.e. the elements in the phase are not required to successfully complete the interaction protocol |
| **Relations** | | | |
| Name | Type | Cardinality | Description |
| preconditions | Milestone | 0..* | The set of milestones that must be reached before the phase can be started |
| postconditions | Milestone | 0..* | The set of milestones that is reached at successful completion of the phase |
| orderedInteraction Structure | Ordered Interaction Structure | 1 | The root (top-most) interaction structure |
| description | Element Description | 0..1 | A description of the phase |
| **Examples (in pseudo concrete syntax)** | | | |

```
<service>
…
 <interactionProtocol xmlns:bsn=”http://www.moonbank.com/BankingServicesNetwork/”>
 …
  <phases>
   <name> Capture Application Details </name>
   <degenerated> false </degenerated>
   <optional> false </optional>
   <postconditions id=”ms534”> … </postconditions>
   <orderedInteractionStructure> … </orderedInteractionStructure>
  </phases>

  <phases>
   <name> Submit Application </name>
   <degenerated> false </degenerated>
   <optional> false </optional>
```

*(continues on next page)*

```
    <preconditions> #ms534 </preconditions>
    <postconditions id=”ms535”>
     <name> Application submitted </name>
     <description>
      <longDescription>
       The application details have been successfully submitted and a receipt acknowledgment has been received.
      </longDescription>
     </description>
    </postconditions>

    <orderedInteractionStructure>
     <occurrence> 1 </occurrence>
     <orderType> Sequence </orderType>
     <elements xsi:type=”interaction:Interaction”>
      <name> Submit Account Opening Request </name>
      <description>
       <longDescription>
        The request with all captured information is sent to the service to start off the account opening process.
       </longDescription>
      </description>
      <involvedRoles> bsn:PROV-1234-9876 </involvedRoles>
      <reverse> false </reverse>
      <exposedAction> #action435 </exposedAction>
     </elements>
    </orderedInteractionStructure>

   </phases>
  </interactionProtocol>
…
</service>
```

## 2.5 Milestone

A **Milestone** indicates the achievement of an important stage in the execution of a service. Milestones provide a way to express the major states that a service will reach during its execution. States are not necessarily limited to "execution states" of a service (e.g.: invoked, running, suspended, or completed), but could, for example, indicate the progress of processing a certain object. In this sense, states have to be understood as specific to the service's application domain. In consequence, it is possible to reach more than one state at the end of a phase. On the other hand, phases might require certain states to be reached before they can start.

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

| Milestone | | | |
|---|---|---|---|
| Attributes | | | |
| Name | Type | Cardinality | Description |
| name | EString | 1 | The name of the milestone |
| Relations | | | |
| Name | Type | Cardinality | Description |
| description | Element Description | 0..1 | A description of the milestone |

| Examples (in pseudo concrete syntax) |
|---|

```
<service>
…
 <interactionProtocol xmlns:bsn=”http://www.moonbank.com/BankingServicesNetwork/”>
 …
  <phases>
  …
   <postconditions id=”ms534”>
    <name> Data captured </name>
    <description>
     <longDescription>
      Through a questionnaire, all relevant information for opening a new bank account is captured. This
      includes customer data (identification and contact details) and options for the selected account.
     </longDescription>
    </description>
   </postconditions>
  …
  </phases>
 …
 </interactionProtocol>
…
</service>
```

## 2.6 Interaction

An **Interaction** represents actual communication between service consumer and service. On a technical level this could translate to calling a Web service operation. On a professional level, it could mean that consumer and provider meet in person to exchange service parameters or resources involved in the service delivery (e.g. documents that are processed by the provider).

Interactions are the lowest-level building blocks of the interaction protocol and expose an atomic point of communication, which when enacted may lead to individual actions being performed. At least one agent partaking in the service's delivery is involved in this enactment of the interaction. This means a communication channel has to be provided and communication with the consumer has to be performed. **Note:** an interaction does not necessarily need to result in a service action (operation) to be performed. For example, a series of interactions might just collect input data from the consumer before one interaction will finally trigger the action.

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

| Interaction | | | |
|---|---|---|---|
| Attributes | | | |
| Name | Type | Cardinality | Description |
| name | EString | 1 | The name of the interaction |
| reverse | EBoolean | 1 | Indicates whether the interaction is initiated by the consumer (false), which is often the case, or by an agent participating in the provisioning/delivery of the service (true) |

| Relations | | | |
|---|---|---|---|
| Name | Type | Cardinality | Description |
| triggeredActions | Action | 0..* | The interaction may trigger a number of actions, i.e. will result in these actions to be performed |
| involvedRoles | Role | 1..* | The set of roles that enact the interaction, i.e. receive input from the service consumer, process it and/or return results |

Examples (in pseudo concrete syntax)

```
<service>
…
 <interactionProtocol xmlns:bsn="http://www.moonbank.com/BankingServicesNetwork/">
 …
  <phases>
  …
   <orderedInteractionStructure>
   …
    <elements xsi:type="interaction:Interaction">
     <name> Personal Details </name>
     <description>
      <shortDescription> Provide personal details </shortDescription>
     </description>
     <involvedRoles> bsn:PROV-1234-9876 </involvedRoles>
     <reverse> false </reverse>
     <exposedAction> #action433 </exposedAction>
    </elements>
    <elements xsi:type="interaction:Interaction">
     <name> Account Configuration </name>
     <description>
      <shortDescription> Provide account options and bank transfer details </shortDescription>
     </description>
     <involvedRoles> bsn:PROV-1234-9876 </involvedRoles>
     <reverse> false </reverse>
     <exposedAction> #action434 </exposedAction>
    </elements>
   </orderedInteractionStructure>
  …
  </phases>
 …
 </interactionProtocol>
…
</service>
```

## 2.7 OrderedInteractionElement

**OrderedInteractionElement** serves as the super type of all elements/concepts that can be part of an ordered structure of interactions.

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

| OrderedInteractionElement | | | |
|---|---|---|---|
| **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 |
| optional | EBoolean | 1 | Indicates whether the ordered interaction element is optional, i.e. not required to successfully complete the interaction protocol |
| **Relations** | | | |
| Name | Type | Cardinality | Description |
| description | Element Description | 0..1 | A description of the ordered interaction element |

## 2.8 OrderedInteractionStructure

**OrderedInteractionStructure** is used to place interactions in a block-structured order.

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

| OrderedInteractionStructure | | | |
|---|---|---|---|
| **Attributes** | | | |
| Name | Type | Cardinality | Description |
| orderType | OrderType | 1 | Determines the type of ordered structure |
| **Relations** | | | |
| Name | Type | Cardinality | Description |
| elements | Ordered Interaction Element | 1..* | The parts of the ordered structure |
| **Examples (in pseudo concrete syntax)** | | | |

```
<service>
…
 <interactionProtocol xmlns:bsn=”http://www.moonbank.com/BankingServicesNetwork/”>
 …
  <phases>
  …
   <orderedInteractionStructure>
    <occurrence> 1 </occurrence>
    <orderType> Sequence </orderType>
    <elements xsi:type=”interaction:Interaction”> … </elements>
    <elements xsi:type=”interaction:Interaction”> … </elements>
   </orderedInteractionStructure>
  …
  </phases>
 …
 </interactionProtocol>
…
</service>
```