Publishing of Interoperable Services and Processes in UDDI – Short Paper

Marcus Spies,
Dpt. Computer Science,
Munich University
Email: marcus.spies@ieee.org

Keith Swenson,
Fujitsu Software Corporation,
Sunnyvale, CA
Email: KSwenson@us.fujitsu.com

Harald Schöning,
Software AG,
Darmstadt
Email: Harald.Schoening@softwareag.com

***Abstract*— This paper presents a requirements analysis and a solution approach to representing workflow processes conforming to the Wf-XML/ASAP standard proposal in a UDDI registry. Wf-XML/ASAP is introduced as a representation of business processes focusing on loosely coupled subprocesses which involve executable and human activities. The UDDI standard as part of the web services protocol stack is designed to hold service relevant information from overall business entities down to technical fingerprints of mostly simple web services. We present a solution approach that allows to use UDDI for registering and discovering service endpoints of Wf-XML/ASAP processes involving complex and dynamic interactions of executable and human activities. We conclude that while such a representation is possible, some of UDDI's structures and relationships need to be extended in order to get an appropriate service registry in interoperable services collaborative infrastructures.**

I. Introduction

Wf-XML/ASAP [1] is a specification originating from the workflow management coalition (WfMC) for interoperability of workflow systems that readily generalizes to heterogeneous service infrastructures. In contrast to execution languages like WS-BPEL [2], Wf-XML ASAP focuses on the operations needed for initiating a service instance given a structured service description on behalf of a client, interacting with a service instance by clients or by service management operations, and running a service that is able to communicate with clients or management operation requests. That is, Wf-XML/ASAP is rather a process administration and management than a process execution specification. This is in line with the WfMC workflow reference model [3] that addresses interoperability on general scale.

If a Wf-XML/ASAP platform is implemented using web services, the operations correspond to interfaces provided by *factory* (creating instances), *observer* (instance clients and management operations), and *instance* (running) services. In a web service deployment, the role of a factory could be played by a single process deployed to a WS-BPEL engine that creates instances automatically upon client invocation. It should be noted, however, that Wf-XML/ASAP adresses also more general settings where an instance can actually be a set of long-running interacting processes (like in government legislation) that can be initiated by different factories (automated or human-task based). The deployment might as well happen in a grid distributed resources implementation as on a WS-BPEL compliant process engine.

Central to Wf-XML/ASAP is also a set of general-purpose interfaces for run time interactions with running instances for management or monitoring purposes. Again, this is most relevant to reconfiguring long-running processes involving complex interactions and human tasks on demand during run time. Since the interfaces in question are defined independently of the current process state, the term *asynchronous service access protocol* (ASAP) has been coined to describe this aspect of the WfMC interoperability specification.

As a side remark we add that despite of its limited impact on the business process languages standardization initiatives, Wf-XML/ASAP is nevertheless supported by several vendors of business process engines with very high market shares like Software AG and Fujitsu Systems.

The basic purposes of the Wf-XML/ASAP specification are to handle

- business processes that can be called via some general client interface – like a web portal page or call center of an insurance company that serve to pass customers to specific automated services or human consultants dedicated to personal insurances,
- collaborative services and business processes in which more or other interested parties than the callers participate on an ad-hoc basis – like financial controlling processes in public administration initiated on a regional level, while a local tax administration and some particular business are the parties involved,
- business processes with ad-hoc components and long durations including intermittencies due to opening hours, limited human resources – like processes in public administration, medical services, or research.

Such processes are sometimes referred to as *agile processes* intending that partners with specific interfaces as well as activities or entire subprocesses might join into a process instance at run time. This is somewhat opposed to the more traditional business process reengineering (BPR) view that assumes processes to conform to schemata entirely defined at design time.

A major requirement for complex service infrastructures involving interoperability of heterogeneous services is the availability of suitable registries that allow to retrieve service definitions and service endpoints (static for factories, dynamic

for instances) in an efficient way. An obvious candidate specification for such a registry is UDDI [4].

The purpose of the present paper is to establish a method for using the universal description, discovery and integration standard UDDI (see [4]) as service registry in Wf- XML 2.0 together with the asynchronous service access protocol (Wf-XML/ASAP, see [1]).

The purpose of a Wf-XML service registry compliant with [1] is to

- Publish metadata about services or business processes
- Allow clients to
  - discover a service
  - retrieve a schema of a service (in a BPMN [5] compliant format)
  - bind to the service

UDDI 3.02 [4] is an obvious candidate for implementing an ASAP/Wf-XML service registry component, since UDDI allows to

- publish metadata about services,
- use references to interface specifications to communicate schemata of service calls,
- provide access points to deployed services.

In addition, UDDI defines a powerful search interface and allows for federated/replicated implementations.

In the present paper we assume UDDI 3.02 as a given standard. However, the extensions that are implied by the requirements for UDDI to enable suitable registry functionality for Wf-XML/ASAP also show some limitations of UDDI's data models and query interfaces. Thus, this paper also is intended to contribute to discussions and initiatives for the further development of the UDDI standard beyond its current version.

II. Related work

For the purpose of the present document, we assume service interface and service binding information to be published in the web service description language (WSDL 1.1, see [6]). Since UDDI is a standard for web services registries, and since Wf-XML specifies services composed to form larger workflows or other business processes, there are only a few additional specifications necessary to represent Wf-XML services that obey the ASAP protocol in a UDDI registry.

A widely adopted proposal for representing WSDL 1.1 information in UDDI can be found in [7]. A representation following the proposals of [7] of essential parts of a WSDL service description is summarized in table I. The column entitled *category bag references* shows which information elements are referred to by suitable entries in the UDDI category bag structure of the representing entity. E.g., a WSDL service is represented as a UDDI businessService entity with references to the service description's namespace and local name in its category bag. This representation is designed to account for the different entities in a WSDL document in a suitable way. The necessary categorization tModels are explained in detail in [7].

WSDL element	UDDI representation	Category bag references
portType	tModel	namespace
binding	tModel	namespace, portType tModel, protocol, transport
service	businessService	namespace, local name
port	bindingTemplate	binding tModel, portType tModel

TABLE I

Summary of essential WSDL 1.1 element representation in UDDI from [7]. For explanation, see text.

A central assumption of the proposal [7] is the separation of interface and implementation information in WSDL along the lines of port types and bindings on the one side, services and their ports on the other. However, that proposal [7] was neither explicitly referred to nor taken up in the UDDI v3.02 [4] specification. On the other hand, the parts of UDDI 3.02 specification that relate to WSDL allow a different grouping of information items than proposed in [7]. In the same vein, the new WSDL 2.0 specification [8] also does not follow the subsumption of binding information into the interface component as suggested in [7]. This subsumption was not in accordance with the usual concept of an interface as in UML.

In the present specification outline, we propose a solution related to [7] in the framework of a UDDI v3.02 registry. Furthermore, as changes from WSDL 1.1 to WSDL 2.0 do not affect the cornerstones of this solution and since the web service definition of the ASAP protocol is in WSDL 1.1 format, we decided to cast it in WSDL 1.1 terminology and structures. In accordance with WSDL 2.0 [8], we use the term interface in the context of a web service not in the sense of [7] but rather as synonym for port type.

III. Component roles in Wf-XML/ASAP compliant interoperable services

We briefly sketch the roles of Wf-XML/ASAP components in retrieving, calling and managing services, see figure 1. For details, see [1].

The ASAP *factory* is a web service responsible for instantiating a business process. On demand by calling clients, this factory creates *instances* and returns an address reference of the process instance to the calling client and/or further *observers*. Still other observers may register for the process instance during process lifetime by subscribing to the process instance in question. We stress that the address reference is a *service endpoint reference* in the sense of the WS-Addressing specification [9] that contains information sufficient to uniquely identify a given service instance and is thus more specific than the general endpoint information contained in the service part of a WSDL document.

A. Registry requirements for interfaces and endpoint references

The Wf-XML/ASAP components of factories, instances, and observers (see section III and the introduction above) are assumed in this paper to be implemented as web services.

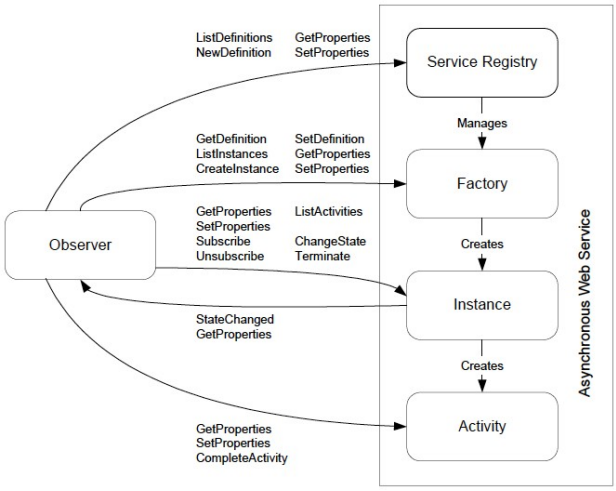

Fig. 1. Wf-XML extension of ASAP, from [1], p. 8. Note that only one observer instance is depicted, while, in reality, several observer instances can participate in one instance of a Wf-XML/ASAP process. A client calling the factory for creating an instance usually implements the observer port type, as well.

Therefore, their roles assumed in performing a service are specified in WSDL as interface information (port type and binding elements in WSDL) and endpoint information (service element in WSDL).

A Wf-XML/ASAP *factory* service can be readily published in a UDDI compliant service registry. Both interface and endpoint information can be easily provided within the usual WSDL information element structures.

As for service *instances*, only the interface information of an ASAP service instance may be published ahead of instantiation of an instance by a factory in the registry, since the endpoint is dynamically assigned by the factory. Since the service instance interface is different from that of the service factory, both roles are described by different port types.

With some services, it may be desirable to publish *only the factory endpoint* in a registry. This is suitable for services with a general client call interface analogous to a call center in human operated processes. Upon a client request, an instance is produced by the factory and its endpoint reference (in the WS-Addressing sense) returned to the client, again in analogy to a personal consulting person assigned for a service request to an insurer by a customer via the insurance company's call center. This person's desk phone extension would otherwise not be available to external customers – similarly, an instance endpoint reference in this case would not appear in a Wf-XML/ASAP service registry. We call such a registry configuration of factories and instances the *combined entity option*.

With other services, it may be necessary to publish *the instance endpoints separately from a factory*. This is suitable for long running or collaborative services in which partners may enter on an ad hoc basis. A simple example would be an online auction where certainly the factory information (including interface information on how to start an auction) would only be published in restricted registry access tables while an instance information including the endpoint reference needs to be published. In other cases likely to occur in public administrative processes, factory and instance information should be available via the registry (again under suitable authentication restrictions and with suitable authorization required for using the services). We call such a registry configuration of factories and instances the *separated entity option*.

Using the separate entity option, it is the responsibility of the factory service to register and unregister instances properly with the registry. This is not foreseen in the current version of Wf-XML/ASAP and is proposed here as an extension.

Wf-XML/ASAP *observers* require yet another port type. This port type subsumes all operations necessary for a client calling an ASAP service. A calling observer who wishes to use an ASAP service must implement the corresponding observer port type for the service with a suitable binding. The endpoints of observers are unknown to a service registry. However, in order to make sure a factory and its instances can communicate with a given observer, this observer must respect the wsdl:bindings used by factory and instance. E.g., in a simple scenario a customer observer endpoint reference (EPR, see [9]) is passed to a factory by a calling service. Some time later, the instance sends a completed request to the observer EPR. This may be, e.g., a rpc-style SOAP message. In order to understand this message, the observer must implement a suitable binding of the completed request/response operation.

As a consequence, in the general case, a subscribing observer must receive not only the instance interface and endpoint information but it must also retrieve the relevant observer interface (port type and binding) information (wsdl:binding) used by the instance to communicate with subscribing observers. This information is instantiation dependent – it is only meaningful for running instances initiated by a given factory.

We add that another option for implementing the aforementioned separated entity option is to have the *registry enroll as an observer to all instances of such services and unregister these instances upon a completion notification*.

B. Service data structures and metadata

In this section, we discuss references to several data structures and metadata information items related to a Wf-XML/ASAP service in a UDDI registry.

A Wf-XML/ASAP service uses pre-specified general purpose state and event enumeration structures, see [10]. These state and event type schemata can be published as specification tModels of category `xmlSpec` (see [4], p.246) containing a reference to the XML schema for the corresponding ASAP states and event types etc. (a part of today's asap.xsd, see the ASAP web page at OASIS).

There are two data schemata for calling and receiving returned data from a Wf-XML/ASAP service [10] –

context data schema this is the schema for the arguments of the process call (like customer number, order reference) to a given port type.

result data schema this is the schema for the port type return structures (like order status, billing information) to the

calling process.

Wf-XML/ASAP services usually are processes themselves (see our examples in section III). Therefore, in addition, a **process definition** in a suitable format should be accessible via the service registry. Usually, the language of the process definition in Wf-XML contexts will be XPDL, see [11], which is a BPMN [5] compliant XML format.

There are two ways to retrieve a process definition that should be supported by a Wf-XML/ASAP implementation and a process registry –

1) process definitions are found by calling GetDefinition on the ASAP factory, see [10].
2) process definitions can be found from the registry using a tModel reference in the businessService category bag. This presupposes another service specific tModel of category `xmlSpec`, which could be further refined, e.g. as XPDL process description.

IV. UDDI Registry Elements for Representing a Wf-XML/ASAP Service

A simple solution to representing component roles of a Wf-XML/ASAP service in a UDDI V3.02 registry [4] in line with the requirements derived can be constructed as follows.

A. Prerequiste general tModel for Wf-XML/ASAP component roles

As a prerequisite, we need to define one generic tModel for categorization of the three Wf-XML/ASAP component roles (factory, instance, observer). This model allows to categorize a UDDI entity as being in one of these roles.

```
<tModel tModelKey="uddi:wfxml.org:asap:roles">
<name>ASAP role</name>
<description>component role according to Wf-XML ASAP
    </description>
<overviewDoc>
   <description>documentation</description>
   <overviewURL>
      http://www.oasis-open.org/committees/...
   </overviewURL>
</overviewDoc>
<categoryBag>
 <keyedReference keyName="uddi-org:types:categorization"
     keyValue="useTypeDesignator"
     tModelKey="uddi:uddi.org:categorization:types"/>
</categoryBag>
</tModel>
```

(For the key value `useTypeDesignator`, see [4], p 249).

This categorization can, theoretically, be added to all defined port type and binding tModels. However, for practical purposes it is most important to add this categorization to category bag of the relevant `bindingTemplate` (requiring UDDI version 3, since in version 2 bindingTemplates do not have a category bag). It is then possible to use the UDDI inquiry API to find a factory of any service just in one step of find_binding using a category constraint.

B. Representing service WSDL key elements

Based on the tModel classification of the ASAP service roles it is possible to construct a UDDI representation for a given Wf-XML/ASAP service in line with [7]. The most important step is to represent needed information for the service's factory role – this is done by defining a businessService for the Wf-XML/ASAP service to be represented and adding to it a bindingTemplate referring to the factory port type and (one of its) binding tModels. The local port name appears in an InstanceParms element of the respective tModel. (This last reference is redundant, as this information is already given in the businessService category bag. It is kept in here to remain compliant with the specification [7].)

For the *combined entity option*, we consider each major WSDL document element as summarized in table I –

port types – define tModels for each of the service's ASAP port types (factory, instance, observer) whose overviewURL element contains references to the appropriate WSDL file(s).

bindings – define tModels for bindings of each of the service's ASAP port types (factory, instance, observer). Each such tModel has a reference to the tModel key of the relevant port type. Note there may be multiple binding tModels for each ASAP component role (factory, instance, observer).

service – define a businessService entry in the registry with namespace according to the WSDL file in keyed reference and with the local port name of the WSDL factory port in another keyed reference.

port – For an ASAP service using the *combined entitiy option* only the factory port is defined in WSDL. Accordingly, as described, this port information tModel is associated with the bindingTemplate of a factory's businessService representation in UDDI.

Finally, add additional bindingTemplates to our businessService for each relevant observer or instance binding (usually just one each) according to the following rules –

- Only the factory `bindingTemplate` has an `accessPoint` with attribute `useType` set to `endPoint`.
- Observer and Instance bindings' `accessPoint` have attribute `useType` set to `wsdlInterface`.
- The `overviewURL` points to the factory service endpoint for a factory binding, to relevant WSDL files for the other ASAP roles.

With this step it is made sure external observers can bind to an instance managed by the businessService factory once they get passed the instance endpoint.

If the *separate entity option* applies, the last step is to be modified – now the instance's `bindingTemplate` has an `accessPoint` with attribute `useType` set to `endPoint`. Note that this is correct since an instance for separate entity services is published only upon existence, and thus the access point holds a correct endpoint reference. Note also that in some cases, the factory service's `accessPoint` might be just published as a `wsdlInterface` (which again can be hidden from public access through a `hostingRedirector` entry, see [4]).

C. Representing service data structures and metadata

Service context and result data schemata (see III-B) will be provided by a business under a service specific URL and can be imported by service wsdl documents. Alternatively, they can be contained by service `wsdl` documents. It is useful to make these schema definitions available via our UDDI registry in the child element `instanceParms` of the `tModelInstanceInfo` parent within the `bindingTemplate` representing the port type implementation of the respective `bindingTemplate`.

In an analogous way, the process description XPDL information (see III-B) as represented by a suitable tModel can be deployed into the UDDI registry under the appropriate Wf-XML/ASP factory bindingTemplate.

As a suggestion for further UDDI releases, it might be desirable to further extend the specification categories available for classifying tModels using the `uddi:uddi.org:categorization:types` UDDI predefined tModel.

D. Usage of the UDDI registry by service clients

In this section, the retrieval of appropriate service endpoints, service interfaces and service metadata for a Wf-XML/ASAP service from a UDDI registry as proposed in our approach is verified by matching the representation of these data in UDDI against the level 1 and level 2 interface calls of the Wf-XML/ASAP specification (for the distinction between levels 1 and 2 interfaces in Wf-XML/ASAP, see [10]).

In the following description lists, $A >> B$ denotes *data A can be retrieved from the registry by searching for UDDI entity B*. Searching implies using the appropriate queries from the UDDI inquiry API (see [4]).

1) Level 1 interfaces and dialogues

   for CreateInstance/RqRs we publish a bindingTemplate for the service factory that contains
   a) the access point (EPR) to the factory >> accessPoint element (useType endPoint or wsdlDeployment)
   b) a reference to the ASAP factory service signature for CreateInstance/RqRs >> service specific tModel available from the bindingTemplate via tModelInstanceInfo
   c) a reference to the applicable context data schema >> instanceParms in the tModelInstanceInfo structure of the bindingTemplate

   for Completed/RqRs we need to publish a bindingTemplate for the observer ASAP role that contains
   a) a reference to the ASAP observer service signature for Completed/RqRs >> tModelInstanceInfo for observer portType tModel
   b) the appropriate binding information of instance EPR >> instance bindingTemplate of the containing businessService
   c) a reference to the applicable result data schema >> instanceParms in the tModelInstanceInfo structure of the bindingTemplate

2) Level 2 interfaces and dialogues

   for ListFactories/RqRs – this is a call to the registry to find all bindingTemplates owning an ASAP factory >> the factory is identified in the registry by a suitable tModel for its portType. This tModel contains the portType's namespace. This namespace, in turn, is part of a keyedReference in the services' businessService UDDI category bag. Finally, these services can be queried for a bindingTemplate related to the relevant portType tModel. As a consequence, all factories for a given portType can be retrieved via the registry structures defined in the present paper.

   for GetProperties/RqRs on Factory – this operation delivers the context data schema needed to issue a CreateInstanceRq and result data schema needed to understand a CompletedRq. >> The appropriate schemata can easily be retrieved from the instanceParms element in the tModelInstanceInfo section of the suitable bindingTemplate.

   for GetDefinition – this operation of a factory delivers a process description in a suitable language (usually XPDL for Wf-XML/ASAP). For calls to such operations to be issued, a client needs access to the following informations from the registry –
   a) the access point to the factory EPR >> accessPoint element (useType endPoint or wsdlDeployment) of the service factory bindingTemplate
   b) a reference to the ASAP factory service signature for GetDefinition/RqRs >> service specific tModel available from the bindingTemplate via tModelInstanceInfo

   for GetProperties on Instance a) the access point information of the instance EPR is obtained either from a response to a CreateInstance or to a ListInstances call
   b) a reference to the ASAP instance service signature for GetProperties/RqRs
   c) a reference to the ASAP status mapping enumeration
   d) a reference to the applicable result data schema >> The appropriate schema can easily be retrieved from the instanceParms element in the tModelInstanceInfo section of the suitable bindingTemplate.

   for SetProperties on Instance a) the access point information of the instance EPR is obtained either from a response to a CreateInstance or to a ListInstances call
   b) a reference to the ASAP instance service signature for SetProperties/RqRs
   c) a reference to the applicable context data schema >> The appropriate schema can easily be retrieved from the instanceParms element in the tModelInstanceInfo section of the suitable bind-

ingTemplate.

for ChangeState/RqRs

the access point information of the instance EPR is obtained either from a response to a CreateInstance or to a ListInstances call

a reference to the ASAP instance service signature for ChangeState/RqRs

a reference to the ASAP status mapping enumeration (TModel)

V. Example

In this example, the travel expense verification and approval service from [10] is used. The code of the following listings has been verified against a apache.org jUDDI registry (that supports UDDI version 2 and some parts of UDDI version 3). In order to keep listings as readable as possible, categorization tModels from [7] have been given UDDI version 3 domain and derived keys (which are based on hierarchical namespaces instead of UUIDs).

We first describe the appropriate registry entry structures, then the use of the registry by clients of the service.

A. UDDI registry entries for the service

First, here is a factory portType tModel as retrieved from the jUDDI database with a call to get_tModelDetail.

```
<tModel tModelKey="uddi:C3880500-A640-11DA-8500-8B60BFBC64E3">
  <name>ExpenseFactory PT</name>
  <description>factory for expense verification service
  </description>
  <overviewDoc>
     <description>wsdl file</description>
     <overviewURL>
        http://travel.com/services/descriptions/expense_fct.wsdl
     </overviewURL>
  </overviewDoc>
  <categoryBag>
     <keyedReference keyName="WSDL Entity type"
       keyValue="portType"
       tModelKey="uddi:uddi.org:wsdl:types"/>
     <keyedReference keyName="portType Namespace"
       keyValue="http://travel.com/expenses"
       tModelKey="uddi:uddi.org:xml:namespace"/>
  </categoryBag>
</tModel>
```

Notes:

1) the tModel name should be equal to the local WSDL port type name according to [7].
2) the portType namespace should be equal to the WSDL document target namespace for the port type.

In a similar way, the tModel of a factory WSDL-binding is built:

```
<tModel tModelKey="uddi:47ACBCD0-A6E7-11DA-BCD0-BE2507793AA6">
 <name>ExpenseFactory SOAP binding</name>
 <description>one binding of factory interface
              for expense verification service</description>
 <overviewDoc>
    <description>wsdl file</description>
    <overviewURL>
       http://travel.com/services/descriptions/expense_fct.wsdl
    </overviewURL>
 </overviewDoc>
 <categoryBag>
  <keyedReference keyName="WSDL Entity type"
            keyValue="binding"
            tModelKey="uddi:uddi.org:wsdl:types"/>
  <keyedReference keyName="portType reference"
            keyValue="uddi:C3880500-A640-11DA-8500-8B60BFBC64E3"
            tModelKey="uddi:uddi.org:wsdl:porttypereference"/>
  <keyedReference keyName="portType Namespace"
            keyValue="http://travel.com/expenses"
            tModelKey="uddi:uddi.org:xml:namespace"/>
  <keyedReference keyName="soap binding"
            keyValue="uddi:uddi.org:protocol:soap"
            tModelKey="uddi:uddi.org:wsdl:categorization:protocol"
                                                  />
  <keyedReference keyName="http transport"
            keyValue="uddi:uddi.org:protocol:http"
            tModelKey="uddi:uddi.org:wsdl:categorization:transport"
                                                  />
  <keyedReference keyName="uddi-org:types"
            keyValue="wsdlSpec"
            tModelKey="uddi:uddi.org:wsdl:types"/>
 </categoryBag>
</tModel>
```

Note that here categories for transport and protocol are added in accordance with [7]. The port type being made available through this binding is contained in the portType reference keyed element of the category bag.

Given the tModels for port types and their bindings, a bindingTemplate can be defined that refers to these entities. This bindingTemplate plays the role of an access endPoint for ASAP factories implementing the given service. As UDDI requires a service key for a bindingTemplate we now define our ASAP service using a domain key and a derived key as follows (assuming we may freely use `uddi:wfxml.org`):

```
<businessService
     businessKey="uddi:travel.com"
     serviceKey="uddi:travel.com:expenses">
 <name>Expenses</name>
 <description>Expense verification</description>
 <bindingTemplates>
  <bindingTemplate
            bindingKey="uddi:E7904100-A6F5-11DA-8100-F3F63E983852"
            serviceKey="uddi:travel.com:expenses">
   <description xml:lang="en">Expense verification service
            factory deployment info</description>
   <accessPoint useType="endPoint">
            http://travel.com/expenses</accessPoint>
   <tModelInstanceDetails>
    <tModelInstanceInfo
            tModelKey="uddi:47ACBCD0-A6E7-11DA-BCD0-BE2507793AA6">
     <description>binding tModel of implemented portType.
            parms contain local port name</description>
     <instanceDetails>
      <instanceParms>expenseServicePort</instanceParms>
     </instanceDetails>
    </tModelInstanceInfo>
    <tModelInstanceInfo
            tModelKey="uddi:C3880500-A640-11DA-8500-8B60BFBC64E3">
     <description>portType tModel of deployed portType.
            parms contain context and result data schema.
        </description>
     <instanceDetails>
        <instanceParms>
            http://travel.com/services/descriptions/expensesSchema.x
        </instanceParms>
     </instanceDetails>
    </tModelInstanceInfo>
   </tModelInstanceDetails>
    <categoryBag>
        <keyedReference keyName="ASAP role"
                keyValue="factory"
                tModelKey="uddi:wfxml.org:asap:roles"/>
    </categoryBag>
  </bindingTemplate>
  <!-- further bindingTemplates for observer, instance
portTypes -->
  <!-- accessPoint for these has useType="wsdlInterface" -->
  <categoryBag>
    <keyedReference
        tModelKey="uddi:uddi.org:wsdl:types"
        keyName="WSDL type"
```

```
            keyValue="service" />
        <keyedReference
            tModelKey="uddi:uddi.org:xml:namespace"
            keyName="service namespace"
            keyValue="http://travel.com/expenses/" />
        <keyedReference
            tModelKey="uddi:uddi.org:xml:localname"
            keyName="service local name"
                <!— from service element in wsdl—>
            keyValue="ExpenseVerification" />
    </categoryBag>
</businessService>
```

Note that the entry of a service specific schema file in the instanceParms element is, strictly speaking, not necessary if we have a full wsdl specification importing these data available via the registry tModel of the port type. However, in the context of a service discovery session it is valuable to be able to access ASAP service context and result data schemata from the service details viewpoint. Therefore, this information has been added here.

Further bindingTemplates for observer and instance interfaces and bindings have to be added with appropriate ASAP role key references in their category bags.

VI. Conclusion – Open issues and future work

The present work shows that UDDI [4] has a data model sufficiently general to accommodate for a service interoperability and collaboration interface specification such as Wf-XML/ASAP. We have also seen the representation to be used is even extensible if additional management / monitoring or governance interfaces are to be added. This is possible due to UDDIs extensible structure employing tModels.

We have also seen some issues to be clarified about Wf-XML/ASAP. One major open issue is the lack of registry dialogue elements in the protocol. We showed in this paper that most of the dialogue elements needed can be adopted from the UDDI find interfaces, but there are exceptions notably in case of registrations of service instances. Another issue about Wf-XML/ASAP is compliance with business rules. E.g., if a running instance has only one observer, should this observer be allowed to send an unsubscribe request? If yes, the instance may as well stop operations. However, it can do so only upon request of its calling factory.

A general issue for service collaborations is authorization. Under many circumstances, requests to set properties of a running instance or one of its activities may lead to inconsistent overall states of business information. At least, within transaction boundaries such requests should not be fulfilled. It is understood that many deployments of Wf-XML/ASAP services will presume a suitable *security infrastructure* to be in place. E.g., an instance reference should only be allowed to be passed to suitably authenticated observers and observers might be authorized to subscribe only to a specific subset of the service data. More generally, there should be a way to distinguish between observers with different view and change permissions. E.g., in a logistics application, we may have a customer observer who is only permitted to track the shipment status of his order, while a tour or an equipment planner is permitted to reroute a container if the shipment is overdue etc. – Security infrastructures like these are implied by the specification documents and need to be defined in precise terms for any specific services deployment.

Different levels of service data and status visibility to different clients poses a challenge to UDDI, as well. The `hostingRedirector` mechanism only helps to ensure authentication in linearly ordered permission sets granted to roles. This presupposition is usually not fulfilled in service collaborations. The support in UDDI for complex authentication and authorization infrastructures is still rather simple since it is defined on a query operation level rather than on a service specific basis. Therefore, access levels to specific service functions or selected service (meta)data are hard to represent and to enforce in UDDI.

A major drawback of the UDDI data model as of today is its inability to represent relationships between services (`businessService` entities) – like dependencies, functional relationships, delegation etc. This not only makes it impossible for a Wf-XML/ASAP service to publish factories and instances in a straightforward way as *services related to each other*. It also prevents semantic relationships between services to be represented and thus hinders exploring a service collaboration infrastructure by registry users.

These and related issues and their implications to using and / or extending UDDI are being investigated.

References

[1] K. Swenson, S. Pradhan, and M. Gilger, "Wf-XML 2.0 – XML Based Protocol for Run-Time Integration of Process Engines," tech. rep., Workflow Management Coalition, 2004.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services," tech. rep., BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems., 2003.

[3] W. M. C. (WfMC), "The Workflow Reference Model," tech. rep., Workflow Management Coalition, Winchester Hampshire, UK, 1995.

[4] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. Dovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. v. Riegen, T. Rogers, K. Sycara, P. Wenzel, and Z. Wu, "UDDI Version 3.0.2," tech. rep., OASIS, 2004.

[5] S. A. White, "Business Process Modellung Notation (BPMN)," tech. rep., Business Process Management Initiative (bpmi.org), May 3, 2004 2004.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Service Description Language (WSDL) 1.1," tech. rep., IBM Corp., Microsoft Corp., 2001.

[7] J. Colgrave and K. Januszewski, "Using WSDL in a UDDI Registry, Version 2.0.2," tech. rep., OASIS UDDI Spec TC, 2004.

[8] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Service Description Language (WSDL) Version 2.0," tech. rep., W3C, 2007.

[9] D. Box, E. Christensen, and F. Curbera, "Web Services Addressing," tech. rep., BEA Systems, IBM, Microsoft, SAP, Sun Microsystems, 2004.

[10] K. Swenson, "ASAP/Wf-XML Cookbook – Updated," in *Workflow Handbook 2005* (L. Fischer, ed.), pp. 257–280, Lighthouse Point, FL: Future Strategies Inc., 2005.

[11] W. M. C. (WfMC), "Process Definition Interface – XML Process Definition Language," tech. rep., Workflow Management Coalition, Lighthouse Point, FL, 2005.