The Role of UDDI in Web Services Change Management

Applies To:
Web Services, Change Management, Web Services Registry

Summary
Web services simplify application integration and implementation of business processes. They provide an extensible framework for application to application (A2A) integration. The simplification of the implementation of business processes is achieved by defining standardized mechanisms to describe, publish, locate, and communicate with other applications. Web service registries provide a systematic way to publish and to find Web services that may be suitable for the development of suchlike business processes. Web service technologies, such as Universal Description, Discovery, and Integration (UDDI) registries, allow doing this in a standardized way. Moreover, a registry that is compliant with the UDDI 3.0.2 OASIS Standard supports not only Web service discovery and publication functions but also infrastructural functions for change management.

Since the management of changes is a critical component in Web services infrastructures, this article outlines the change management related features of UDDI 3.0.2 in more detail. It discusses the role of a service registry focused on Web service change management from the Web service provider as well as from the Web service consumer perspective. Taking the UDDI 3.0.2 specification into consideration it identifies the change management mechanisms that can be provided by such a registry as well as its deficiencies.

By: Ruediger Bachmann

Company: SAP AG

Date: 28 Dec 2005

Introduction
The simplification of the implementation of Web service based business processes and application integration can be achieved by service registries which basically provide three functions. A company’s IT service provider publishes information about its Web services in the service registry. This information can be discovered by organizations, which intend to integrate such advertised services into their own processes. The obtained information is used in business processes of an organization. Universal Description, Discovery, and Integration (UDDI) is a technology that is offered by many solution providers to achieve exactly this kind of registry-based interaction. To sum up, there are the following types of interaction.

- Publish/Unpublish/Update (service provider ↔ service registry)
- Find/Browse/Discover (service consumer/requester ↔ service registry)
- Invoke/Bind (service consumer/requester ↔ service provider)

A key aspect of any IT strategy is change management. Change management for Web services is critical to establish and maintain the integrity of the service throughout its lifecycle. In the context of Web service change management there is still no universally accepted definition of how Web service interactions can be implemented to achieve a lean and effective change management process. The UDDI 3.0.2 specification already addresses several Web service change management requirements, but deficiencies still remain.
Before analyzing the web service change management requirements and how well the UDDI 3.0.2 specification addresses them, let's first review some fundamentals.

**The UDDI Service Registry**

A UDDI registry can be used by anyone who is authorized and who wants to make information available as well as by anyone who is searching for that information. It is a registry for service providers and services and can be accessed by suitable APIs. The information stored in a UDDI registry is typically of four different types:

- businessEntity — a description of a company, an organization or a service provider
- businessService — a non-technical information about services
- bindingTemplate — the representation of a Web service, the service access information
- tModels — technical metadata (transport, protocol, categorizations, structuring, etc.)

There are additional data structures stored in a UDDI registry, but for the purposes of this article, it is sufficient to understand the role of the four data types above (for further details see [1]).

Concerning Web services a **businessService** groups a set of Web service endpoints and maps to a WSDL service. The service access information is contained in the **bindingTemplate** which maps to the WSDL port. A special type of **tModel** points to the service description which could be a WSDL document (for further details see [2]).

**Interactions with the UDDI Registry**

The basic interactions with a UDDI registry are depicted in Figure 1.
Service providers describe themselves and publish this information in terms of *businessEntities* to a UDDI registry. Additionally the Web services they provide are published based on *businessServices*, *bindingTemplates* and *tModels*. On the consumer side human actors or client applications search for suitable Web services which fit into their business process. The search procedure can be based on various criteria as outlined in **UDDI 3.0.2**. Finally, a service can be used by accessing its interface description at the service provider. A Web service consumer generates a Web service client based on this interface description (WSDL), configures it (e.g. set up of authentication settings) and integrates it into the application in concern\(^1\).

### Changing the Web Service

After a Web service has been deployed and is running on the IT infrastructure of the Web service provider, it may be consumed by Web service consumers. Once this has been done and the Web service has been running for a while it will often change in some regard. This can be caused by a number of reasons. Some of them like bug fixes or changing the security configuration are addressed in [3]. There may be Web service clients who fail invoking Web services after such changes. To avoid this kind of failures, a robust change management should be established. For that reason, **UDDI** version 3 provides the Subscription API Set, which is a mechanism that allows Web service consumers to subscribe to certain search criteria or directly to certain UDDI entities in order to get informed about changes in the registry.

These subscription APIs are:

- `save_subscription`
- `get_subscription`
- `delete_subscription`
- `get_subscriptionResults`
- `notify_subscriptionListener`.

The first three provide create, read, update and delete functions concerning the subscription to the occurrence of changes, deletions or the creation of new entities. The `get_subscriptionResults` API returns registry data pertaining to a particular subscription and the `notify_subscriptionListener` API facilitates asynchronous notification.

If subscribers are interested in receiving information about changes using the Subscriptions API Set they first have to register this interest. A registration allows them to track new, changed or deleted entries for *businessEntities*, *businessServices*, *bindingTemplates*, *tModels* and *publisherAssertions*. The registration is carried out by calling the `save_subscription` API. Information about the concerned UDDI entities a subscriber wants to subscribe to can either be provided directly, i.e. by means of a list of entity keys, or indirectly, i.e. by means of search criteria, similar to the find_xx APIs of the Inquiry API set. The `subscriptionFilter` is the crucial argument which actually determines the scope of the subscription to the registry records. Furthermore the

\(^{1}\) Depending on the underlying architecture this procedure may differ. For instance, when developing a BPEL process the definition of the process may be carried out first based on the interface descriptions of the contained Web services. The generation of the appropriate Web service clients may be done implicitly in an integrated build and deployment task dependent on the given process platform.
The Role of UDDI in Web Services Change Management

subscriber specifies within the arguments of the `save_subscription` API in which way changes will be tracked. If the optional argument `bindingKey` is populated with information a notification will be delivered to the specified notification listener. The `bindingKey` specifies the `bindingTemplate` which defines whether the notification is carried out via eMail or via the `notify_subscriptionListener` API. If this optional information is not specified the subscription and notification mechanism has to be established by the use of the `get_subscriptionResults` API. Thus there are two notification mechanisms:

- Synchronous requests
- Asynchronous notifications

Using synchronous requests, subscribers issue a synchronous request to the UDDI registry based on various filter criteria which can be specified in the subscription. They use the `get_subscriptionResults` API to obtain information on activity in the registry which matches their subscription preferences. It is a clock-driven monitoring process which is carried out periodically at registered time intervals. This process does not depend on the occurrence of changes. The periodicity of requesting changes depends on the setting on Web service consumer side. Figure 2 briefly depicts the basic course of action of synchronous requests embedded in an overall process starting from the publication.

![Figure 2: Synchronous change tracking](image-url)
Asynchronous notifications facilitate an event-driven monitoring of changes (Figure 3). The triggering event corresponds to the invocation of the `notify_subscriptionListener` API. The subscription model for asynchronous notifications allows the subscriber to impose time restrictions that specify subscription validity period and notification interval. The (asynchronous) notification API consists of a notification listener that is a ready-to-run callback Web service that is implemented by the subscriber as a subscription listener service and is invoked by the notification service implemented by the UDDI registry. It contains the data that changed since the last call of the `notify_subscriptionListener` for a particular subscription.

Both mechanisms do not need to be used separately from each other. When a notification is missed the `get_subscriptionResults` API can be used alternatively to obtain the subscription data. Appendix C of the UDDI 3.0.2 specification provides some subscription examples.

We have already seen that the scope of the subscription is specified by the `subscriptionFilter` as an argument of the `save_subscription` API. This scope determines the content of the `subscriptionsResultsList` which provides the information about new, changed or deleted entities. Concerning the definition of changes in the specification of the Subscription API Set, technically speaking, it is stated that entities are considered to be changed if the `modifiedIncludingChildren` element of the `operationalInfo` element of an entity has been changed. Furthermore section 3.8.2 of the UDDI 3.0.2 specification states that if a contained entity is changed...
the `modifyIncludingChildren` element of the ` operationalInfo` element of the containing entity is changed (valid for saving, deleting, or move entities). The example in Figure 4 illustrates this interaction.

![Figure 4: Definition of Changed Entities](image)

If the `businessService` changes its enclosing `businessEntity` is considered to be changed because there is an ` operationalInfo` element which points to this `businessEntity` with its `entityKey/businessKey` and contains a `modifiedIncludingChildren` element that changed because of the change of the `businessService` element. Since the containing relation is valid for `businessEntities` and `businessServices` a change of contained entities can result in the population of the `subscriptionsResultsList` even if the changed element has not been explicitly specified in the `subscriptionFilter`. For instance a change of a `bindingTemplate` causes a change of the `modifyIncludingChildren` elements of its containing entities. Therefore a subscriber will be made aware of a change of a `bindingTemplate` even if the `subscriptionFilter` has restricted the subscription only to the containing `businessEntity`. This provides the possibility of “collecting” changes to components of a “higher” entity, which may reduce the number of subscriptions and notifications and reduce the amount of the communication traffic.

**Conclusion and Aspects for Further Discussion**

The subscription and notification mechanism of the Subscription API Set already provides powerful means to monitor and to be notified of changes of a subset of the UDDI registry data. These functions may provide the basis for a sound and holistic change management process in the field of Web service change management. This field is still at its beginning and there are still many questions left to be answered. We will complete this article with a consideration of such aspects.

1. **Deprecations features.** This information could indicate to the Web service consumer that the service is only available up to a certain point of time in the future.

2. **Meta data about changes.** Additional information about the change of a record of the UDDI registry concerning compatibility, affected layers (design time, runtime or configuration entities), changes of the underlying implementation or of the Web service interface could enable the Web service consumer to decide whether a change must be taken into account immediately – for instance by regenerating or reconfiguring the Web service client – or whether the old Web Service client can still be operated.

3. **Correlation between arbitrary entities in the UDDI.** Changes of Web services occur in manifold ways. New entities in a UDDI registry may relate to already existing entities. For example, a logical change
of a Web service may be carried out by means of creating a new Web service based on new businessService, bindingTemplate and tModel entities. A subscriber to the entities relating to the “old” Web service may not necessarily be notified because there is no relationship between the two Web service images in the UDDI registry.

4. **Compound notifications and subscriptions** reduce communication traffic and complexity. We have already addressed the possibility of propagating change notification to other entities by the use of the modifyIncludingChildren elements. This simplifies subscription at the same time, because it is possible to subscribe to more coarse grained entities and to be notified of changes of contained records. It may be possible to generalize this concept to arbitrarily chosen and registered entities in the UDDI registry.

5. **Change descriptions.** Not only metadata about changes but also the concrete content of the changes, their meaning and the consequences for consumers may be helpful for Web service consumers.

Future articles will examine several of these aspects in more detail.

**References**


**Author Bio**

Ruediger Bachmann is a member of the SAP Netweaver Industry Standards team. He joined SAP in 1997, starting as application developer in the IBU Utilities. Contact: ruediger.bachmann@sap.com.