# Unit 6. Service Oriented Architecture

**Outcomes Summary.** You will learn

- How different WS can be coordinated to work together in systematic, standard ways.
- Why the concept of service architecture is interesting for developing large business DSs.
- How various WS standards extend the basic WS to facilitate important aspects of large business DSs such as security, resources, addressing and notification, in particular:
  - how UDDI helps in the discovery and management of services

**Context.** In the previous unit we have learnt the basics of how Web services can enable individual remote nodes to communicate in a simple request/reply manner. (We will learn more complex and more flexible ways to program Web services in the coming units.) We observed that Web services support openness and heterogeneity through detailed and unambiguous remote interface descriptions. In this unit we will think beyond the simple scenarios of a handful of services and clients designed to work with each other. Web services are intended to be used in the context of massive and volatile systems spanning many organisations with hundreds of service providers and users. In such systems a single task often requires numerous service invocations and the cooperation of multiple nodes.

## 6.1 Introduction to SOA

Service oriented architecture (SOA) is a way to design and implement large DSs, in which the basic unit of communication is the invocation of remote services. A service suitable for use within SOA should be:

- available on demand in the long term;
- concisely yet fully described in a *standard* way, consequently
  - easy to connect to and use in any common programming environment;
  - having confined and predictable effects;
  - using standard formats for data exchange;
- accessed in a stateless request-response manner (ie the history of previous requests and responses is not explicitly referred to except via well-described resources).

RESTful services, such as those we have seen in the previous unit, are very good candidates for use in a SOA except for one aspect: we have described these services systematically but informally, ie not using any standard format. A standard format is useful when we want the description to be automatically processed by a computer program, eg in order to automatically generate some repetitive parts of the client or server implementation. For example, if we are implementing a server or client using Jersey, we could perhaps use the standard description to generate a proxy class for the client and/or JAX-RS-annotated Java classes with resource and locator method stubs for the server. There is such a standard format for describing RESTful services, called Web Application Description Language (WADL). An example WADL is in Fig. 6.1. Nevertheless, its use is not very widespread because SOA tends to be implemented using more complex Web services that are described using the Web Service Description Language (WSDL), which we will learn in some detail in the following unit.

Services expose processes (usually business processes) and clients use them to implement applications. The typical textbook example of a SOA system is a network comprising and supporting travel agents. In this system, there are services eg for booking flights, hotels and car hire. Then there are agent applications that use these services to implement more sophisticated holiday-package services to its clients.

Business use of SOA as opposed to distributed objects or *ad hoc* RPC is justified by a promise of:

- easier reuse of services for multiple purposes;
- better adaptability to changing business environment and available technologies;
- ability to integrate new and legacy systems;
- ability to cheaply setup e-business links across the World.

### 6.1.1 Overview of WS standards

In typical SOA scenarios, there are issues that are not addressed by the Web services as we saw them in unit 6, for example:

- *service publishing and discovery*
  Example standard: *Universal Description Discovery and Integration (UDDI)*
  Eg (automatically or manually) search for all car hire Web services that offer pickup at Heathrow and implement an interface, which is described in a WADL or WSDL document that my program knows how to do car hire booking booking with.
- *service choreography*, ie describing sequences of service invocations
  Example standard: *Business Process Execution Language (BPEL)*
  Eg specifying that one has to confirm a booking made with some Web service within 10 min and pay for it using a specific payment Web service within 2 days.
- *management of shared stateful resources*
  Example standards: *WADL, Web Services Resource Framework (WSRF), Web Services Resource Transfer (WS-RT), WS-Addressing*
  Eg specifying which booking is being confirmed or paid for.

```
<?xml version="1.0" encoding="UTF-8"?>

<application
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://research.sun.com/wadl/2006/10"
>
    <grammars>
      <include href="Translator.xsd"/>
    </grammars>

    <resources base="http://localhost:3000/translator/">

        <resource path = "texts">

            <method name="POST">
                <request>
                    <representation mediaType="text/plain" />
                </request>
                <response>
                    <representation mediaType="text/plain" />
                </response>
            </method>

            <resource path = "{textId}">

                <param name="textId" style="template" type="xsd:int"/>

                <method name = "GET">
                    <response>
                        <representation mediaType="text/plain" />
                    </response>
                </method>

                <resource path = "translation/{langFrom}:{langTo}">

                    <param
                       name="langFrom" style="template" type="xsd:string"
                    />
                    <param
                       name="langTo" style="template" type="xsd:string"
                    />

                    <method name = "GET">
                        <response>
                            <representation
                                mediaType="application/xml"
                                element="TranslatedText"
                            />
                        </response>
                    </method>
                </resource>

            </resource>

        </resource>

    </resources>

</application>
```

Figure 6.1: An example WADL document specifying a RESTful interface to translator

- *notifications* of changes in stateful resources
  Example standard: *WS-Notification*
  Eg subscribing to receive notifications when a booking status changes, eg when it expires or had to be cancelled.
- *distributed transactions*, ie coordinated update of multiple resources
  Example standards: *WS-Coordination, WS-Transaction*
  Eg booking and confirming several connecting flights, each through a different Web service — if one fails, all do.
- *service life-cycle management* (eg deploying, upgrading, decommissioning) and monitoring
  Example standard: *Web Services Distributed Management (WSDM)*
  Eg finding out which flight booking services are currently less busy; starting replicas of a travel service to cope with high season load.
- *reliability* of messaging beyond TCP/IP
  Example standards: *WS-Reliability, WS-ReliableMessaging*
  Eg automatically resuming after long network blackouts.
- *security* (eg authentication, encryption, permissions management)
  Example standards: *WS-Security, WS-Trust, WS-Policy*
  Eg identifying rogue servers, testifying in favour of trusted services so that others know they can trust it too.
- *service usage contracts* (eg payment, performance, booking, penalties for failures)
  Example standard: *WS-Policy*
  Eg arranging exclusive access to services for some time and paying for it; publishing costing schemes for a service.

While in principle all these aspects can be implemented using the basic standards (as we have shown for notifications and RESTful services), it is essential that people agree on standard ways of doing so, so that the benefits of SOA are preserved. There are WS standards for all of the above aspects, as indicated in italics with each bullet point. Nevertheless, some of the standards are not yet widely implemented and are undergoing a maturing process that involves occasional substantial revisions. Also, it is common to find competing WS standards for the same purpose, each advocated by different group of companies or standards-defining bodies. It remains to be seen which standards will prevail in the long term.

On the technical side, it is interesting to note that many of these standards build on the basic standards, eg they are supported by XML schemas and WSDL specifications of messages and port types, or they specify specific headers that should be included with SOAP messages (see the following units).

## 6.2 Service discovery using UDDI

Assume one needs to extend a travel agent application to support hotel booking in Hawaii. How can one discover a suitable hotel-booking WS and its WSDL specification? It can be done manually by searching the Web for pages that advertise such

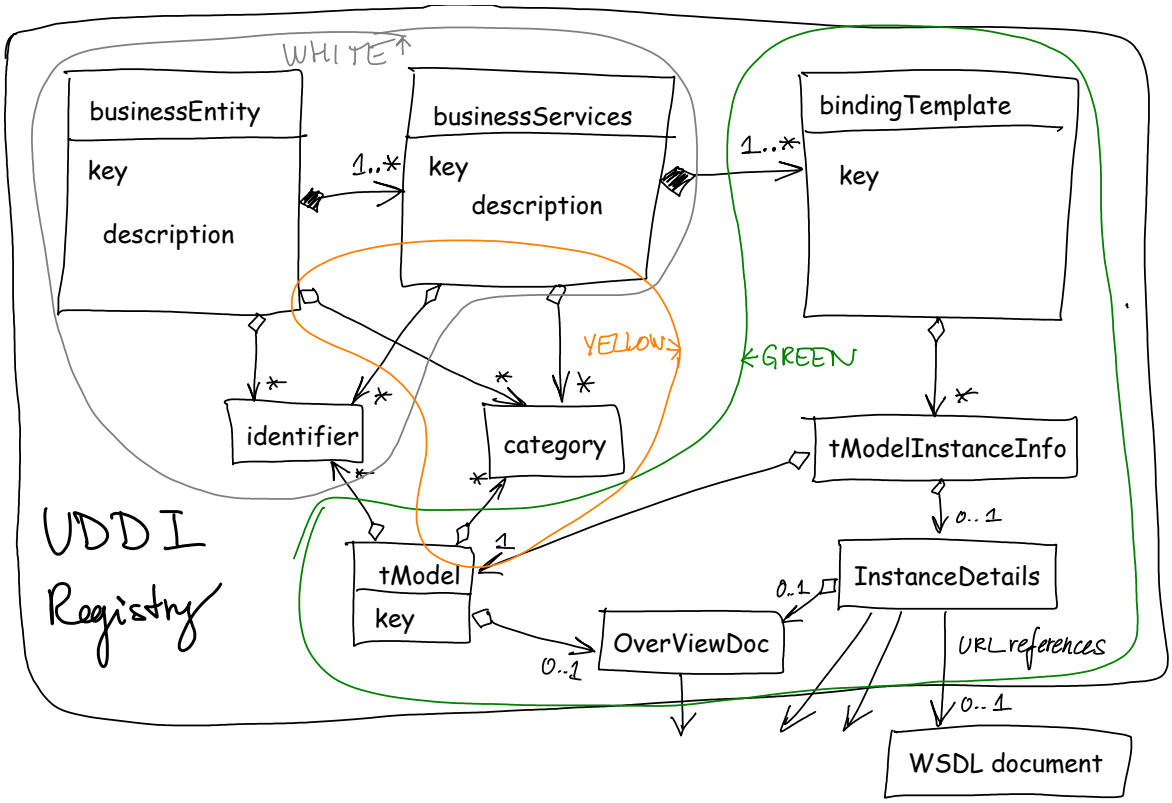


Figure 6.2: Main UDDI data structures

services and when successful, inserting a URL of the discovered WSDL document into the application. Nevertheless, this is quite unreliable and inefficient and is almost impossible to automate. Ideally, the discovery and application adjustment would happen automatically when an operator indicates their wish to add the option to book hotels in Hawaii. With an automated discovery, the system will be able to take advantage of any newly introduced suitable services.

UDDI specifies how to interact with a registry of Web services that can facilitate both manual and automated discovery of services. A UDDI-compliant registry is itself a Web service and it allows:

- service providers to publish information about their services;
- service users to
  - browse a catalogue of services,
  - search this catalogue by various criteria and
  - obtain all technical information about the services they need to determine whether it is suitable and how to use it.

In an analogy to traditional business directories, the data stored by a UDDI registry is often classified as follows:

- *White pages*: businesses and their services listed by their *name*.
- *Yellow pages*: businesses and services classified by keywords and business taxonomies.
- *Green pages*: technical descriptions of individual services, using standards such as WSDL.

The data is stored in a UDDI registry in a very flexible structure (see Figure 6.2) to accommodate all kinds of human- and computer-friendly information.

A *business entity* stores information about a particular service provider. It mainly keeps a human-readable business description that can be searched by keywords. There are also links to identifiers such as the registered names or business registration codes used by the "white pages" listings. The categories facilitate "yellow pages" style search and correspond to elements of well-established taxonomies, such as the codes used by libraries to classify books or the common country codes, postcodes etc.

A business entity contains one or more *business services*. A UDDI business service is actually a family of similar Web services provided by the business entity. These families of services can also be categorised and associated with unique identifiers (such as URIs or registered product codes).

Each Web service port is identified by a UDDI *binding template*, which serves as a portal to the "green pages" section, leading to the WSDL and other technical descriptions of the service.

Each service is described by a number of instances of *technical models* (tModels). A tModel is a generic description of one aspect of a service. A tModel is generic in the sense that it does not relate to a specific service, ie it does not specify a service URL. There are tModels specifying various aspects of services, eg type of connection such as HTTP or email or the resources/operation avialable in the service usign partial WADL or WSDL (partial because it lacks the URL of the service). Other WS standards can be used to formally specify tModels eg types of resources associated with the service, or security, choreography, reliability aspects.

Technical service specifications such as WSDL documents are not held directly at the registry but only pointed to by the registry.