Unit 8. Web Services with Resources

Outcomes Summary. You will learn

- About the latest standards for describing WS with stateful resources.
- To develop and deploy WS with resources using the Globus Toolkit.
- How WS resources integrate with other WS features, namely notification.

Further Reading: Globus Toolkit 4 online tutorial 1.3, 3--8.

Context. In Unit 6 we have identified a number of issues and features that are common to many WS systems and therefore benefit from being standardised. We have also stated a few characteristics of services in service oriented architectures, one of them being that subsequent interactions with a service should be stateless except where a state is kept by the service in "well-defined resources". In this unit we focus on this idea of a well-defined resource and investigate the current state of associated WS standards.

8.1 Managing state in WS

Stateless and stateful services. Until recently, it has been a well-established SOA practice to insist that each service is stateless. For example, a language translation service is usually stateless, because each request is self-contained — it specifies the text to be translated, the source and target languages and any other necessary parameters to guide the translation. While the stateless translator is free to keep records of its activity (ie maintain a state that transcends multiple client requests), each request would makes perfect sense even if all previous requests are forgotten and its response should be independent of any previous requests. For example, with a **stateless** translator:

- When a client needs to translate one text into French and German, it *cannot* send one request for French and then a second requests that says something like "translate the text from the *previous request* into German". (The service could have an operation to translate one text to multiple languages in one request, though.)
- When a client asks twice for the same text to be translated to the same language, the translator *cannot* respond with a message that amount to something like "look at what I sent you *last time*".

The reason for preferring stateless services is that they are much easier to specify and it is easier to deal with failures.

While stateless services are sufficient to implement quite a lot of common business logic, it turns out that there are cases where stateful services are unavoidable. For

example, a stateless flight booking system would be unable to let clients make reservations and confirm them later with subsequent requests. In scenarios such as these developers often created stateful web services in an ad-hoc manner, eg generating custom identifiers for flight bookings and sending these identifiers to clients so that clients can refer to specific bookings. This solution was not acceptable to one important group of Web service users, namely the developers of Globus — an important scientific Grid infrastructure. (We will visit Grids more systematically in the following unit.)

The Globus community decided to build a sophisticated distributed middleware on top of WSDL-based Web services but they needed plenty of stateful services, warranting their standardisation. They developed their own stateful WS standard and implemented it in Globus version 3 and later re-worked the standard in cooperation with the wider WS community and implemented it in Globus version 4 with Java and C interfaces. The latter standard is called Web Services Resource Framework (WSRF). It has also been implemented in other contexts, eg in the Apache Muse project (with a Java interface) and also within the .NET environment.

By 2006, Web services in industry tended to use a simpler standard called WS-Transfer. Nevertheless, WS-Transfer lacks important features, such as direct access to subcomponents of structured resources and resource lifetime management. These features were already present in WSRF but the industrial groups did not want to switch completely away from WS-Transfer. In a bid to reconcile, the WSRF and WS-Transfer communities lead by Microsoft, Sun and IBM developed a new standard called Web Services Resource Transfer (WS-RT), which is formally an extension of WS-Transfer with features of WSRF, in particular WS-ResourceProperties and WS-ResourceLifetime. As there are not many usable implementations of WS-RT available at the moment, we will focus on WSRF in our study.

8.1.1 WS resources

Recall that RESTful services are fundamentally stateful but at the same time enjoy similar benefits as stateless services, namely: scalability, failure tolerance and simplicity. The core of WSRF can be seen as a fusion of stateless WS and RESTful resources, thus retaining much of their benefits.

The main idea of WSRF is to separate a stateful service into one stateless service and a REST-style resource that keeps the state. Such a pair is termed **WS-Resource**. The resource is itself a WSDL-specified service but its operations conform to a simple uniform pattern allowing the stateless service to eg download an XML representation of the whole resource, upload a new representation of the resource and also manipulate each internal component of the resource in a similar fashion. For example, a WSRF flight booking service keeps no information about individual bookings directly, but is associated with a resource for each booking. Every time a request is made concerning a particular booking, the correct booking resource is specified inside the request. This is illustrated in Figure 8.1.

Unlike in RESTful systems, the client cannot access the resources directly. The resources are private to the main service and the clients only get to know an identifier for

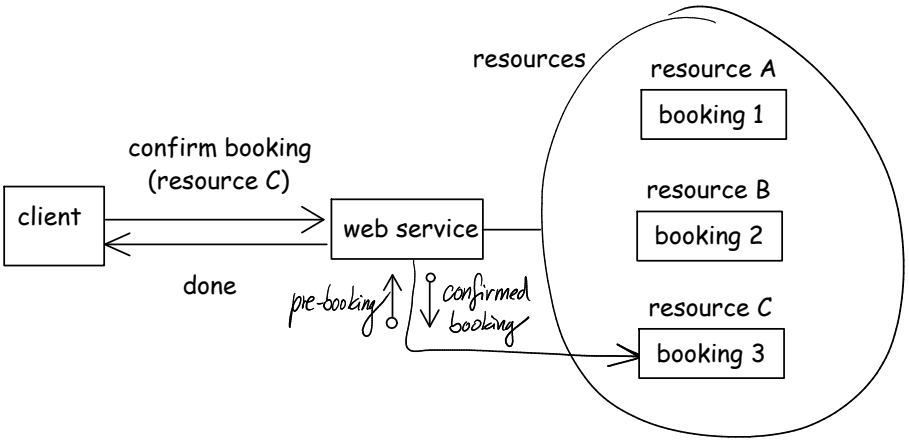


Figure 8.1: Usage of WS-Resources

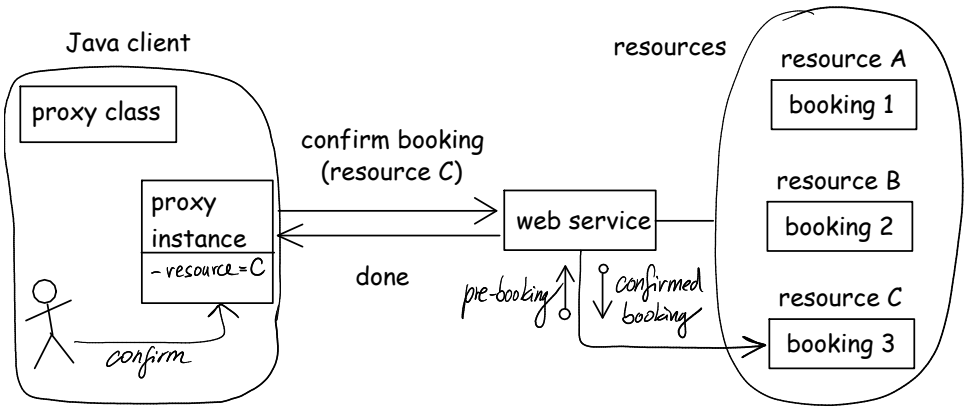


Figure 8.2: Usage of WS-Resources via a typical Java interface

their allocated resource. The clients cannot do anything with such an identifier except send it as part of a request to the stateful service.

This setup is similar to the way Java classes and objects work. A WSRF service is like a Java class and its resources like instances of this class. A user of a Java object holds a reference to that object but typically cannot do anything with the object except for calling its methods (assuming all fields in the object are private). In the same way a user of a WSRF resource has some remote reference to the resource and all it can do is to apply some operations to the resource using the owning Web service. All operations of the service work with a particular resource in the same way as Java instance methods work with a particular instance object.

To match this intuition, when using a WSRF service in Java, one usually has a proxy class whose instances are automatically mapped to individual resources. The methods of the proxy object automatically insert a reference to the associated resource to each request as illustrated in figure 8.2.

8.1.2 Addressing resources

How exactly does the client specify a resource in the SOAP requests? To reinforce the service-resource pairing, the resource ID is always specified as part of the service URI. For example if the resource ID is `C` and the base service URI is:

```
http://a.com/services/Math
```

then the WS-Resource URI is:

```
http://a.com/services/Math?res=C
```

This URI is included in the SOAP envelope as a header. This header uses a general-purpose format for specifying the destination of the SOAP message, following another standards called WS-Addressing. If we use `wsa` as a prefix for the namespace of XML elements defined in WS-Addressing, the SOAP message will look something like this:

```
<soap:Envelope ...>
   <soap:Header>
        <wsa:To>
          http://a.com/services/Math?res=C
        </wsa:To>
        ...
   </soap:Header>
   <soap:Body>
        ...
   </soap:Body>
</soap:Envelope>
```

The purpose of WS-Addressing is to specify how to represent "remote references" to Web services in XML, and in particular in SOAP headers to aid message routing. (Routing is usually handled by HTTP but there are cases where it is desirable to do it at the SOAP level.) A remote reference to a Web service is called an **Endpoint Reference** (EPR). An EPR to a WSRF service has to specify a WS-Resource and therefore includes the `ref` attribute as shown in the URI above. In XML outside SOAP headers an EPR should be wrapped by WS-Addressing elements as follows:

```
<wsa:EndpointReference>
    <wsa:Address>
        http://a.com/services/Math?res=C
    </wsa:Address>
</wsa:EndpointReference>
```

8.1.3 Managing resources

It remains to be shown how a client can initiate the creation of a resource and how it obtains its reference. For this purpose, a WSRF service is usually paired with a *factory service*. This service has a standard port type with one operation `createResource` that takes no parameters and returns an EPR of the new WS-Resource, as illustrated in Figure 8.3.

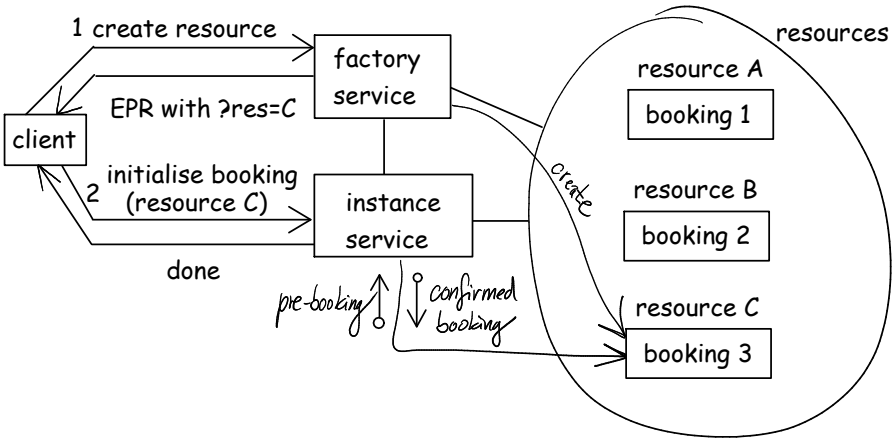


Figure 8.3: Instance and factory services forming a WSRF service

8.1.4 Accessing resource components

Apart from defining the notion of WS-Resource, WSRF comprises several subordinate standards for common operations that involve WS-Resources. This subsection is about one of these standards.

In most cases a resource will be a record of several components (called properties) in the same way as Java objects typically contain several fields. It is often desirable to expose some of these properties to clients. The WS-ResourceProperties standard is a component of WSRF specifying how to expose properties to clients. This standard defines the following WSDL port types, each containing *one* operation whose name coincides with the name of the port type:

- `GetResourceProperty`
- `GetMultipleResourceProperties`
- `SetResourceProperties`
- `QueryResourceProperties`

A service that wishes to use one or more of these port types has to "extend" them, ie include the operations. As WSDL 1.1 does not support extending a port type, the standard's operation definition is usually repeated inside the service's own port type definition. Nevertheless, the message and element definitions are imported and reused from WSDL files provided by the standard.

8.1.5 Notification of resource changes

WS-Notification is a standard closely linked to WSRF and depends on WSRF but is not a formal part of WSRF. WS-Notification supports a fairly flexible model of publishing and subscribing to various types of events expressed as "topics". Formally, the basic version of the standard defines the following port types:

- `NotificationConsumer` (ie event listener)

  - contains operation `notify` called by a producer

- `NotificationProducer`
  - while a producer's primary role is to send notifications, it does not need any own service to do that; nevertheless, it has a service that acts as a factory of subscriptions
  - contains operation `subscribe` called by consumers
    * has a parameter `ConsumerReference` which is an EPR of the consumer;
    * has other parameters to specify eg topic, precondition
    * returns an EPR of a `SubscriptionManager` WS-Resource that represents the new subscription

- `SubscriptionManager`
  - is a WSRF service in which each resource corresponds to one *subscription*, ie a relationship between one consumer and one producer established by a call to the `subscribe` operation
  - this service is used by both notification consumers and producers
  - has an operation `Destroy` to terminate the subscription
  - has other operations to eg pause a subscription or schedule a expiry time for the subscription

In the context of WSRF, WS-Notification is used to enhance WSRF services so that its clients can subscribe for notifications of state changes in a particular WS-Resource. For example, a client of a flight booking service may want to be notified whenever a booking it has created has been altered (in case it happened as a consequence of another client's request). In this case, the topic of the subscription is the whole resource. It is even more common to specify a property of a resource as a notification topic and the `notify` message then contains both the old and new value of the property so that the client has convenient access to full information about the changes as soon as they happen.

Figure 8.4 shows a scenario where client 1 subscribes for notifications about any changes in its booking (stored as resource B). The booking service uses a subscription manager to create a subscription WS-resource. Then a change in the booking is initiated by client 2 and client 1 is being notified. Before this happens, the booking service checks that the subscription is active, in case it has been paused or destroyed by client 1. Finally, client 1 destroys its subscription to notifications about its booking.

8.2 Implementation of WSRF services using Globus

The goal of these notes is not to describe in all details how to implement and use a WSRF service. Nevertheless, we will review parts of an implementation to help clarify the WSRF concepts further. The programmer tools for WSRF are still evolving. Currently one of the easiest ways to create and deploy a WSRF service is via a local

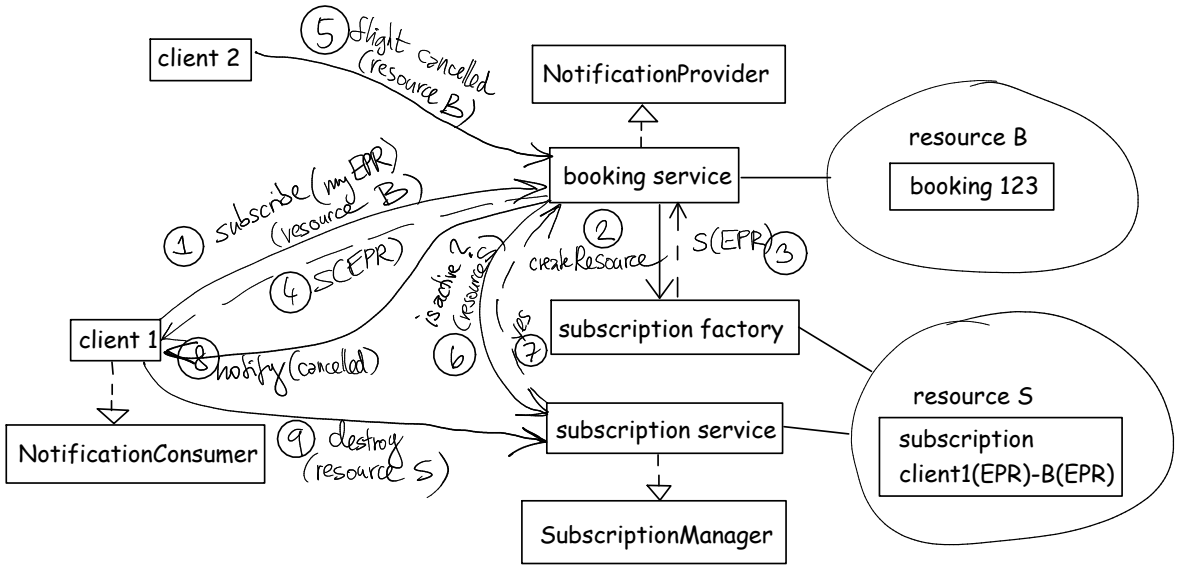
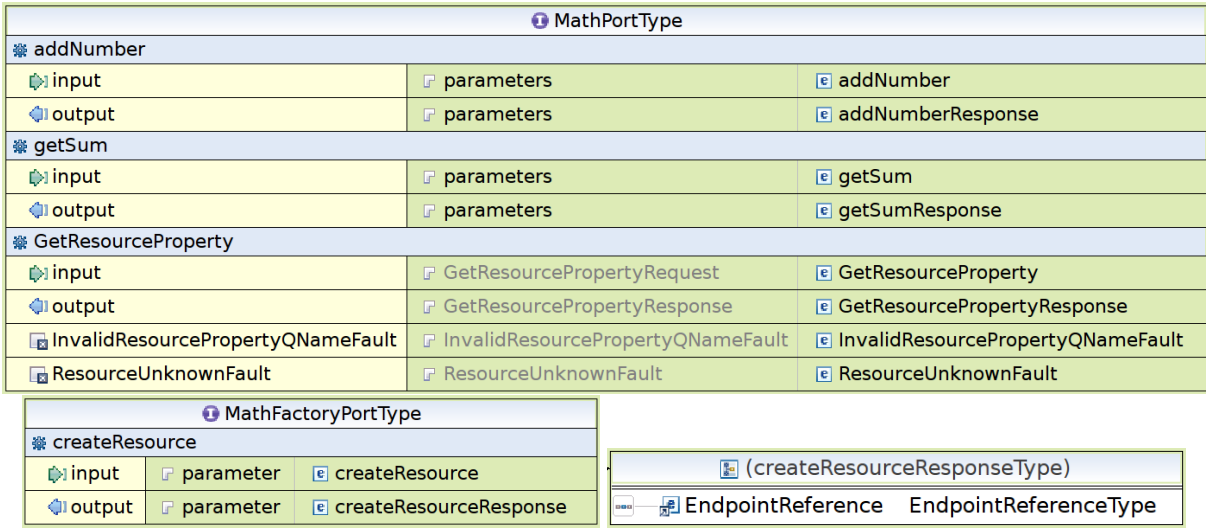


Figure 8.4: An example use of notification in a WSRF service



MathPortType		
addNumber		
input	parameters	addNumber
output	parameters	addNumberResponse
getSum		
input	parameters	getSum
output	parameters	getSumResponse
GetResourceProperty		
input	GetResourcePropertyRequest	GetResourceProperty
output	GetResourcePropertyResponse	GetResourcePropertyResponse
InvalidResourcePropertyQNameFault	InvalidResourcePropertyQNameFault	InvalidResourcePropertyQNameFault
ResourceUnknownFault	ResourceUnknownFault	ResourceUnknownFault

MathFactoryPortType		
createResource		
input	parameter	createResource
output	parameter	createResourceResponse

(createResourceResponseType)	
EndpointReference	EndpointReferenceType

Figure 8.5: Structure of GDT generated WSDL port types for the Math service

installation of Globus Toolkit 4 (GT4). The online tutorial by Sotomayor and Childers (2005) is a good further resource to guide you if you want to try it for yourself and get a good practical understanding of WSRF and Globus.

The Marburg Ad-hoc Grid Environment (MAGE) Grid Development Tools (GDT) make the development of GT4 WSRF services even easier by a good integration with Eclipse. The extracts shown here will be based on the use of GDT.

With GDT, one can have all skeleton and proxy classes generated in a similar fashion as with Axis for ordinary Web services. Nevertheless, GTD works in a code-first fashion, ie it does not allow the programmer to write their own WSDL specification, it generates it from a specially annotated Java class. This annotated Java class is often the only file a service programmer has to edit.

```
@GridService
 (name = "Math",
  namespace = "http://localhost:8080/wsrf/services/MathService",
  targetPackage = "wsrf.math",
  serviceStyle = "SSTYLE_FACTORY",
  resourceStyle = "RSTYLE_MAGE",
  operationProvider = "GetRPProvider",
  loadOnStartup = false,
  hotLoadable = false,
  securityDesc = "[]")
public class Math {
    @GridMethod
    public void addNumber(int n)
    {
        sumSoFar += n;
    }

    @GridMethod
    public int getSum()
    {
        return sumSoFar;
    }

    @GridAttribute
    private int sumSoFar;

    public int getSumSoFar()
    {
        return sumSoFar;
    }

    public void setSumSoFar(int sumSoFar)
    {
        this.sumSoFar = sumSoFar;
    }
}
```

Figure 8.6: A GTD annotated class for a stateful service for adding numbers.

An example GDT annotated class is shown in Figure 8.6. The Java annotations are used by the GDT tools to correctly generate WSDLs and skeletons for both the instance and factory services. Additionally, a single proxy class is generated, which makes use of both of these services to create WS-Resources as illustrated in Figure 8.2.

Figure 8.5 shows some aspects of the generated WSDLs. Notice how the WS-ResourceProperties GetResourceProperty operation definition has been copied into the service port type. It is possible to specify in the annotated class that the service should be bundled with further operations from the WS-ResourceProperties standard and GDT can also make the service into a notification provider according to WS-Notification.