WSRF and Triana

Ian Wang

16th May 2006

1 Introduction

In this tutorial we describe how to create and access WSRF resources using Triana. To use this tutorial we assume a full installation and basic knowledge of Triana. Triana is open-source and can be downloaded from `www.trianacode.org`. A user guide is also available from this site, the Web service section of this guide may be of particular help.

2 Terminology: Contexts and Endpoint References

The WSRF defines conventions from accessing and manipulating stateful resources (WS-Resources) via Web service interactions. Within the message exchanges involved in invoking a Web Service, the resource that is being referred to is identified by the inclusion of an endpoint reference (WS-Address) in the message header. The inclusion of this endpoint reference (EPR) gives context to the message exchange, in that the invocation will occur in the context of the identified resource.

In Triana, we refer to an endpoint reference as a context, and this context can be applied to multiple Web service interactions. The reason for this is that the Triana context framework is intended to be extendible to context types other than WSRF, for example a WS-Security security context could be attached to a Web service (or any other type of contextual service for that matter). For the rest of this tutorial we will generally talk about contexts, however in terms of WSRF this should be understood to be referring to endpoint references (EPRs).

3 WSRF Example

In figure 1 we illustrate a simple WSRF-based workflow in Triana. The basic process of this workflow is to create a stateful counter resource using a *CounterFactory* Web service. Once this stateful counter has been created, a *add* Web service is invoked in the context of this counter in order to add to the counter.

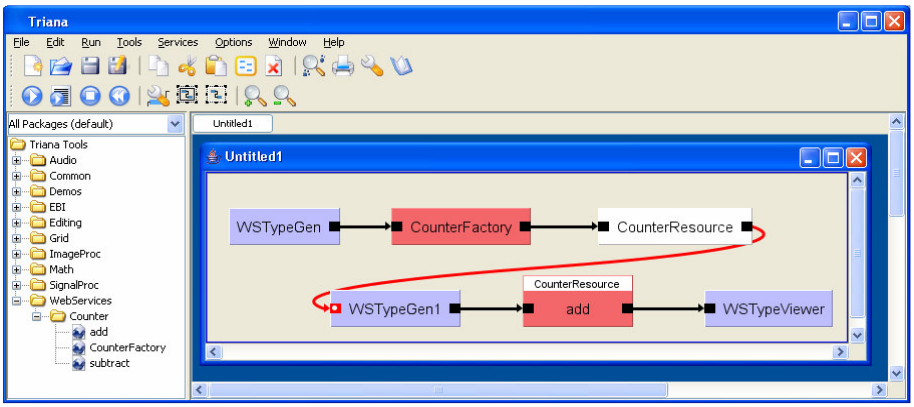

Figure 1: Example WSRF workflow in Triana. Uses a *CounterFactory* service to create a *CounterResource*, and then invokes an *add* service in the context of that resource.

The example workflow above can be broken down into two separate processes; first creating the counter resource, and then applying an operation in the context of that resource. The following tasks are used in creating the counter resource:

- **WSTypeGen** This tool automatically generates a form for providing the input required by *CounterFactory* Web service. This tool is available in the `Common.WebServices` tool box.
- **CounterFactory** This is a factory Web service for creating counter resources. The response from this service includes an EPR referring to the resource that was created.
- **CounterResource** This is an instance of the `Common.Context` tool. It receives the response from the *CounterFactory* service and associates the EPR contained in that response with a context name. In this example the context name is *CounterResource*.

In this example there is no data dependency between the output from the factory service and *add* operation. However, it is required that the *add* task is not enacted until the counter resource has been created. The dependency between the two parts of the workflow is therefore a control dependency rather than data based (as indicated by a red arrow). We discuss how to specify control flow in Section 8.

The following tasks are used in the second part of the workflow, invoking the *add* service in the context of the counter resource:

- **WSTypeGen1** A second instance of WSTypeGen is used to automatically generate a form for the input required by the *add* operation.

- **add** This is a WSRF enabled Web service for adding to a counter resource. In the example in it is invoked in the context of *CounterResource*, the name associated with the EPR returned from the factory service. This means it will add to the counter we created in the first part of the workflow.
- **WSTypeViewer** This tool is used to display the response from the *add* operation.

In the following sections we describe how to create WSRF workflows such as the example given above.

4 Importing Services

WSRF enabled services can be imported into Triana in exactly the same way as standard Web services. To import a WSRF enabled service use the *Services→Import Service* menu option, and insert the WSDL location for the service into the dialog displayed. The operations available on the imported service will appear in the tool tree in the `WebServices` folder. WSRF enabled services can also be discovered from a UDDI repository using the *Services→Discover Services* menu option

5 Creating Named Contexts

WSRF does not specify how resource instances are created or how the EPR for a resource is acquired by the user. Typically however some form of factory service is employed to create a resource and return an EPR for that resource to the user. Triana supports this form of resource/context creation through the `Context` tool which is located in the `Common` tool box.

The `Context` tool is used at design time to associate a name with the resource reference returned from a factory service. This name can then be attached to other task within the workflow to indicate those interactions take place within the context of the named resource. This function can be seen in Figure 1, where a `Context` tool is used to associate the name *CounterResource* with the output from the *CounterFactory* service. This name is later associated with the *add* operation to indicate that this operation takes place in the context of the *CounterResource*.

Within a workflow, either double-clicking on a `Context` task or right-clicking and then selecting *Properties* allows the context name to be set. In the dialog that is displayed, there is also an option to save the context to file. By default the context will be saved in the *∼/.TrianaV3Resources/Contexts* directory.

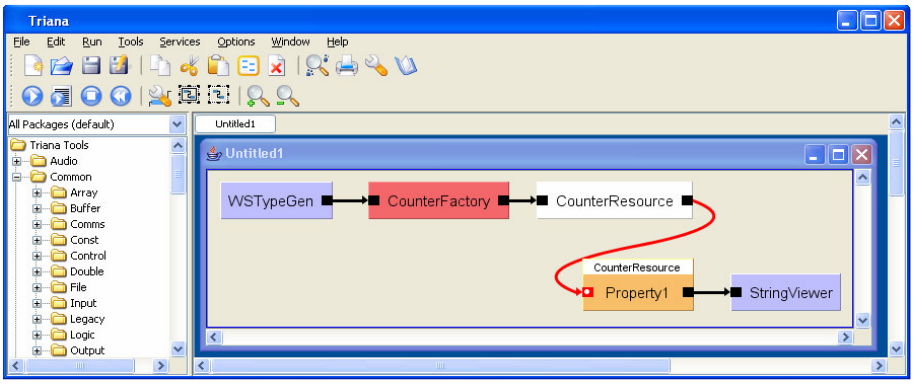

Figure 2: Example workflow for retrieving the value of a WS-ResourceProperties (*Property1*).

6 Associating Contexts with Workflow Tasks

Once a named context has been specified, that context can be associated at design time with other tasks in the workflow. Typically this will be associating a context with a Web service operation. In order to associate a context with a Web service or other contextually aware workflow task, either double-click on that task or right-click and select *Set Context*. A list will appear allowing the selection of one or more contexts to apply to the task. If the required context is stored on file then it can be loaded into the context list by clicking on the *Add* button.

7 Setting/Retrieving Resource Properties

The WSRF allows resources to have resource properties (WS-ResourceProperties) and defines standard operations for setting and retrieving these properties (SetResourceProperties, GetResourceProperty etc.). Although Triana could wrap these standard operations as Web service tasks and use them directly in the workflow, it is more visually intuitive to encapsulate the property as a workflow task. With this model, directing input into a WSRF property task sets the value of the property, while output from the task assumes the value of the property. When a WSRF-compliant Web service is imported into Triana, WSRF property tasks for each resource property are automatically imported into the tool tree.

An example workflow for retrieving the value of a WS-ResourceProperty is given in Figure 2. In this workflow the value of the *Property1* property associated with the *CounterResource* is retrieved and output to a *StringViewer*. Note that in this example, the cable into *Property1* is a control link as there

is no data dependency between the resource factory and property retrieval sections of the workflow. If the cable had been a data cable then the input value would have set the value of *Property1*.

It should be noted that the output from a WSRF property task is not compatible with the `WSTypeViewer` tool. If you wish to simply view the output then the `Common.String.StringViewer` tool can be used.

8 Specifying Control Flow

In Triana, workflow execution is generally data flow based, with the output from one task causing the execution of the next task in the workflow. However, Triana does support control flow between tasks where there is no specific data dependency. This is particularly useful in WSRF based workflows where there is often no data dependency between the output from the factory service and the input into the following tasks.

To enable a control dependency between tasks, the task that will execute when a control signal is received requires a trigger input node. Any input then sent to that node will trigger the task to execute (the value of the input is ignored).

The method for adding a trigger node to a task depends to the type of task. For Web service and WSRF property tasks, simply right-click on the task and select *Add Trigger Node*. For other tasks right-click on the task and select *Node Editor*. Within the node editor select the *Input Params* tab, and then *Add* to add a parameter node. This will cause the *Add Parameter* dialog to appear from where you should select *Trigger Node Only*.