GT4 C Core Design

Design of WS-Resource Framework in C for GT 4.0

Table Of Contents

1 Web Service Implementation and Design

In the Globus WS C architecture, a web service is represented on the server side as a dynamically loadable module (AKA: shared object or library) that gets loaded by the container (the umbrella process) as needed. This allows entire services to be added to a container dynamically, and provides complete scope separation between services. See the GT4 WS C Design Doc for further info.

The reusable operation provider design specified in the C architecture allows us to implement a set of operation providers as components that provide WSRF base functionality. The GetResourceProperty operation provider, for example, provides service functionality for resources properties that implementers can derive from with their own services.

2 Resource Discovery

Resources in C are maintained by the service module in a registry. This allows services to control and limit access to resource instances from other services.

Because a service is invoked through interfaces defined by the service description, the resource id taken from the properties field of the WS-Adressing component of the request message cannot be passed directly to the service implementation. Instead, a service module defines a variable global to the service module, which can be used as a key for storage of the resource id. Globus provides an abstraction layer to thread-specific data handling, which allows the service implementation to gain access to the resource id in both threaded and nonthreaded environments. For example, a service might provide an implementation of the 'add' operation, where the interface would look like:

```
int my_service_add(xsd_int              value);
```

The service module calls this function during dispatching, which may need to access the associated resource. To do so, it must first access the resource id using the function:

```
const char * globus_service_get_resource_id();
```

The resource id is set as thread specific data in the service module during dispatching of the `my_service_add` service operation. Once the service operation has been invoked, the resource id is determined from the service module's global thread key (unique to that thread), which has been set previously. The following function is responsible for resolving the key to the actual Resource instance.

```
globus_result_t globus_resource_find(
const char *                    id,
globus_resource_t *             resource);
```

Because a resource instance may be accessed by multiple operation invocations at once (or at least, within the same request timeframe), each call to `globus_resource_find` increments a reference counter on the actual `globus_resource_t` instance. This prevents calls to `globus_resource_destroy` from destroying the instance while its still in use. The reference counter is decremented for each call to `globus_resource_find` by the service module once the service operation has returned.

3 RESOURCE FACTORIES

A WS-Resource must include a factory operation that creates the resource and responds with the Endpoint Reference for the resource. The factory operation will need to call:

```
globus_result_t
globus_resource_create(
    const char *                    id,
    globus_resource_t *             resource);
```

This function creates the resource instance, adds it to the resource registry keyed on id, and returns the instance to the user. Additional resource specific internal data should be added to the resource via a call to:

```
globus_result_t
globus_resource_set_resource_specific(
    globus_resource_t               resource,
    void *                          data,
    globus_destroy_func_t           destroy);
```

4 ASSOCIATING A RESOURCE WITH A WEB SERVICE

In C, all resource association is done through an implicit approach, in that the service implementation must obtain a reference to the resource through the provided helper API. The service does not maintain association with a particular resource from one call to the next, instead, the resource is automatically returned to the resource bank or home once the service call has completed.

5 CLIENT API

The client-programming model is similar to the programming model in GT3 and designed to handle the complexities of interaction from the user. For example, the client will be able to pass a WS-Addressing EndpointReferenceType instead of a Grid Service Handle to the client stub and the stub will add the appropriate endpoint reference to the SOAP header during serialization of the outbound message.

6 SERVICE API

The helper API available to the service consists of a set of functions that act on resource instances:

```
typedef struct globus_resource_s        globus_resource_t;

globus_result_t
globus_resource_create(
    const char *                        id,
    globus_resource_t *                 resource);

globus_result_t
globus_resource_find(
    const char *                        id,
    globus_resource_t  *                resource);

globus_result_t
globus_resource_delete(
    const char *                        id,
    globus_resource_t *                 resource);

globus_result_t
globus_resource_destroy(
    const char *                        id);
```

This API allows the service implementer to create new resources (using `globus_resource_create`) and return the new resource identifier in a resource factory response message. Also, the Destroy operation provider will likely call `globus_resource_destroy` on the resource.

Internally, the resource properties are maintained as a hashtable keyed on the property's name. Users of the API (service implementers) only have access to the resource properties through a resource property API, which is defined in the following sub-section.

6.1 Resource Properties

The resource properties for a resource instance can be access through the following API:

```
typedef struct globus_resource_property_s globus_resource_property_t;

globus_result_t
globus_resource_get_property(
    globus_resource_t                   resource,
    xsd_QName                           name,
    void **                             property);

globus_result_t
globus_resource_set_property(
    globus_resource_t                   resource,
    xsd_QName                           name,
    void *                              property);

globus_result_t
globus_resource_create_property(
    globus_resource_t                   resource,
    xsd_QName                           qname,
    globus_serialize_func_t             serialize,
    globus_deserialize_func_t           deserialize,
    globus_initialize_func_t            initialize,
    globus_destroy_func_t               destroy,
    void *                              property);

globus_result_t
globus_resource_delete_property(
    globus_resource_t                   resource,
    xsd_QName                           qname,
    void *                              property);

globus_result_t
globus_resource_destroy_property(
    globus_resource_t                   resource,
    xsd_QName                           qname);
```

The service implmenter must add the resource property values defined in the WSDL of the service with default values to each resource that they create. Service implementers may also choose to add a resource property type to a service definition. Both of these are done with the globus_resource_create_property function, which requires serialization and intialization functions. Not adding the resource properties defined in WSDL to the newly created resource invalidates the resource. If the type of the resource property is defined in the WSDL schema, then these functions will be automatically generated, otherwise the service implementer will have to implement the serialization and initialization functions himself. The prototypes for these functions are as follows:

```
typedef void (* globus_xsd_type_destroy_func_t) (void *);

typedef globus_result_t (* globus_xsd_type_init_func_t) (void **);

typedef void (* globus_serialize_func_t) (
    xsd_QName                           name,
    void *                              instance,
    globus_message_handle_t             message);

typedef void (* globus_deserialize_func_t) (
    xsd_QName                           name,
    xsd_QName                           type,
    void *                              instance,
    globus_message_handle_t             message);
```

6.1.1 Client Handling of Resource Properties

On the client the list of resource properties returned in the response messages of GetResourceProperty, GetMultipleResourceProperties, and QueryResourceProperties include a sequence of xsd:any types. By default, the client bindings for a given portType definition generate a table that maps resource property names to initialized `globus_xsd_any_info_t` structures, which contain a `globus_deserialize_func_t` callback for the given resource property. This mapping table must be passed in to the resource properties operations to allow valid deserialization of the responses. Resource properties that aren't defined by the mapping table get deserialized with the default mapping, which turns properties into DOM elements. Each mapping table can be modified at runtime to specify the resource properties to expect, as well as allow the default mapping to be overridden, so that deserialization of resource properties can be user defined.

7 OPERATION PROVIDERS

7.1 WS-ResourceProperties

The request message for the SetResourceProperties operation requires handling of the xsd:any type. As described in the GT4 C WS Design Doc, Each service module provides a mapping table for the resource properties defined on the associated resource. The mapping table is initialized to contain default mappings from resource property name to `globus_serialize_info_t` instances. Modification of the table happens either via a call to `globus_resource_create_property`, or through the API provided. Resource Properties which are added dynamically should be included in the Resource Properties document service discovery queries.

The GetResourceProperty, GetMultipleResourceProperties, and QueryResourceProperties operation providers should also support accessing the Endpoint Reference as a resource property.

7.2 WS-ResourceLifetime

The `SetTerminationTime` operation provider is implemented using the globus event handling API. Specifically, `globus_callback_register_oneshot` is called with a callback that waits until the termination time has expired and then calls `globus_resource_destroy` on the resource.

8 IMPLEMENTATION PLAN

A first-pass implementation of GT4 in C will only include the following features:

- xsd:any support
- WS-Addressing Handler
- Resource Properties support
- Resource Lifetime support
- NotificationConsumer operation provider

The following are specific components that will not be supported in GT4.0:

- Xpath querying of resource properties. Specifically, the QueryResourceProperties operation will fault. (Although libxml provides an Xpath 1.0 compliant implementation)
- Notification/Topics Support.