BioMOBY Asynchronous Services

Specification

(v 2.4.2)

Contributions:

Enrique de Andrés
José María Fernández González
Johan Karlsson
Sergio Ramírez
José Manuel Rodríguez Carrasco

Coordinators: Oswaldo Trelles, David González-Pisano

Node GN-5: Integrated Bioinformatics, University of Málaga
Node GN-2: Bioinformatics and Proteomics, CNIO
Node GN-7: Visual Genomics and Proteomics, CNB

23 April 2008

INSTITUTO NACIONAL
DE BIOINFORMÁTICA

1. - Preliminaries

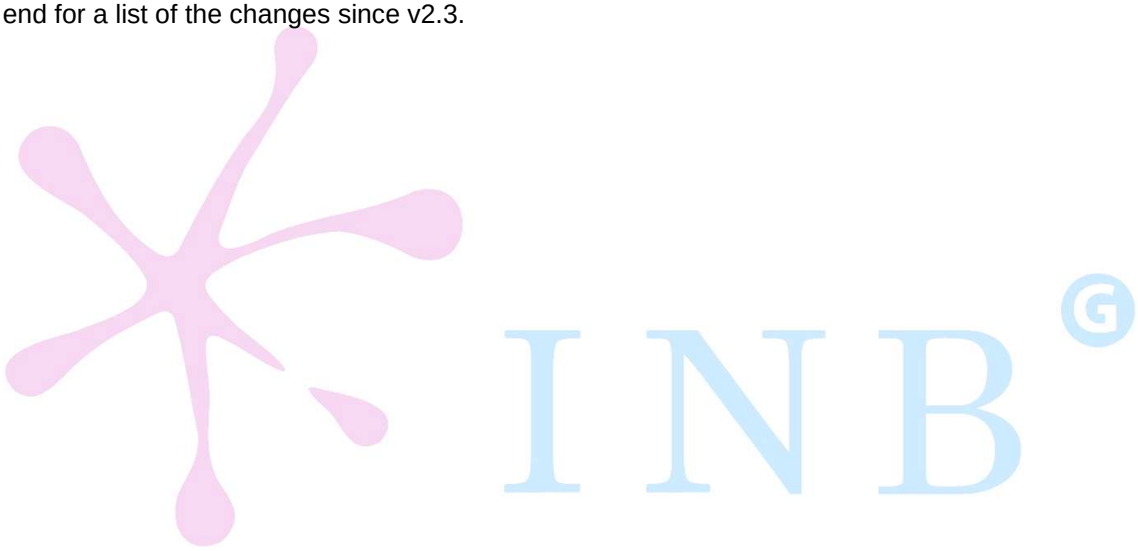
Throughout this document, we will assume that the reader is already familiar with the basic concepts of BioMOBY. For more information about BioMOBY, please study the official documentation[1].

This document contains the specification from INB[2] of how to deal with asynchronous services in BioMOBY. This specification is a result of discussions during the INB Meeting in Málaga (July, 2005) with the participation of Martin Senger and Edward Kawas, and in the INB mailing lists. During further discussion in the "MOBY-dev" mailing list, a suggestion was put forward to investigate if it was possible to base the messaging on an OASIS standard called WSRF.

The aim of the specification is to contribute to the *standardisation* of asynchronous service calls in BioMOBY. A new extended set of service calling operations and defined XML messages are detailed.

The main motivation for the specification is to facilitate the implementation of "long-running" services, i.e. services that demand enough computational resources to need more than a few minutes to compute the result.

The main lines of this specification were ratified in October 2006. Some incorrect examples of input/output from the operations have been corrected in this version. Please see errata in the end for a list of the changes since v2.3.



[1] http://www.biomoby.org

[2] Instituto Nacional de Bioinformática (INB), Spain, http://www.inab.org

2- Current BioMOBY specification

In the **MOBY-S 0.86.3** version of the BioMOBY API, services present their interfaces as Simple Object Access Protocol (SOAP[3]) RPC.

A MOBY compliant service (registered as having the service protocol "moby") is one that uses only object/service classes defined in the MOBY Central registry, presents its service interface via SOAP, and registers this service interface in MOBY Central. In the coming months it will be expanded to allow the registration of non-MOBY SOAP services, as well as CGI services, but this will not affect the API described below for MOBY-SOAP services**.**

Table 1 - Current BioMOBY definition of "MOBY compliant service"

A service can be called through a single procedure (operation), by using the name the service was registered with in the MOBY Central catalog.

After retrieving a service description (currently in the form of a simplistic, but legitimate, WSDL document) from MOBY Central, client programs will subsequently communicate directly with the service provider, first by sending a request, in the form of an input message and then, if all goes well, by receiving an output message. The communication takes the form of a very simplistic SOAP RPC call: the name of the remote procedure call is the same as that when it was registered in MOBY Central. The URI (uniform resource identifier) looks like a URL (uniform resource locator), but is subtly different. Where a URL is the address of a document on the Internet, a URI is an abstract identifier which allows the service to be uniquely identified. At this time, the URI for this procedure call is always http://biomoby.org, as in: http://biomoby.org/#your_procedure_call_name This is regardless of the URI for the service provider! This is useful because the same service might be available from several providers. If they all use the same URI, then a computer (or human) can infer that they are equivalent, and swap one for the other, based on availability, or other criteria.

Table 1 - Current BioMOBY specification for SOAP RPC calls

HTTP is the usual transport protocol. Although not forced by the BioMOBY specification, most of BioMOBY service providers install a web server that handles the SOAP requests.

The most straightforward paradigm is to have a single SOAP server running as a CGI script, and this listener hands-off requests to the appropriate code module as requests arrive.

Table 2 - from "Constructing MOBY-S Compliant Services"

Several problems related to long-running services have been identified in BioMOBY v0.86.3 specification. The same transaction is used to request the execution of a service and to wait for the results. This causes the system to remain occupied and non-responsive in the meanwhile. Also, it is a *de facto* standard to set a connection timeout both in server and clients, closing the socket being used for client-service communication. Often the connection timeout is no more than a few minutes, making it impossible to call long-running services.

[3] http://www.w3.org/TR/soap/

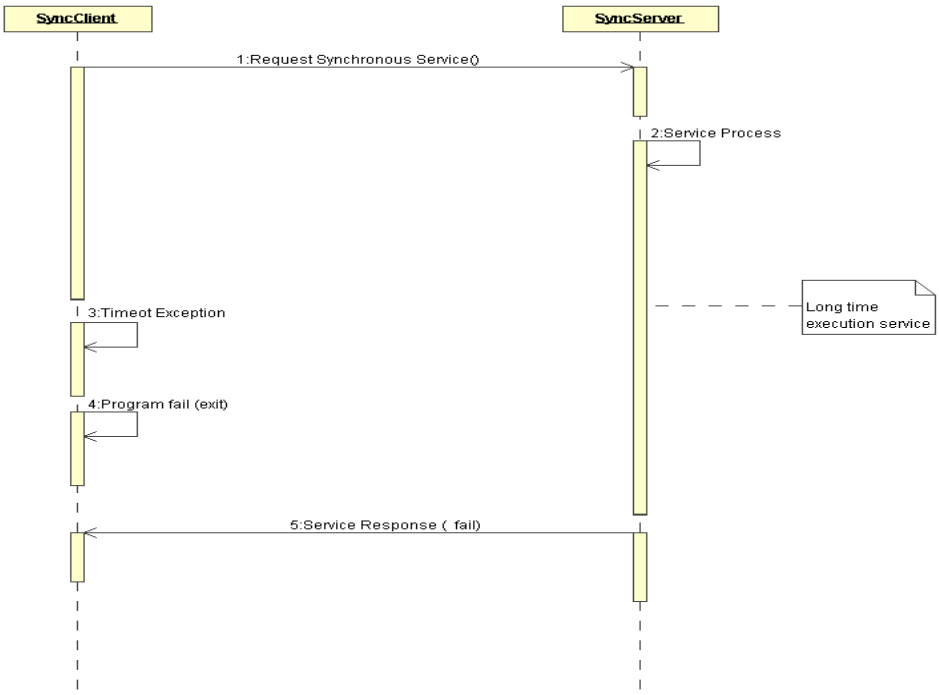


Figure 1 – Illustration of the problem with long-running services in the current BioMOBY specification

3- Specification of asynchronous services/clients

To enable calls to long-running services, it is better to dedicate a separate connection for each step (thereby using an asynchronous communication model).

In this chapter, we specify how to add such support to the BioMOBY specification. We have selected a polling approach. This requires that the service maintains a state. Therefore, we will use an OASIS standard called WSRF to help implementation of state-aware services.

Before we continue, let us describe some concepts that we will use throughout this document:

Batch-call (or session):

This is a service invocation and corresponds to one MOBY/mobyContent message.

Job:

A job is one of the multiple service execution requests that belongs to a service invocation, and corresponds to one mobyData message.

3.1. WSRF - Web Services Resource Framework

In this section we aim to provide a brief, general overview of WSRF. Those interested in details of WSRF and the operations defined there are invited to study the documentation[4].

WSRF is an OASIS standard that can be used to implement interoperable state-aware web-services (WS). WSRF uses the W3C standard WS-Addressing to achieve transport-neutral addressing for web-services. The concept of End-Point-References (EPR) is central in WS-Addressing. An EPR is used to refer to a specific service instance.

End Point Reference (EPR)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      ...
      <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
         <wsa:Address>http://myserver.com/MyService</wsa:Address>
         <wsa:ReferenceParameters>
            <rpimpl:CartId>S1</rpimpl:CartId>
         </wsa:ReferenceParameters>
      </wsa:EndpointReference>
      ...
   </soap:Header>
   <soap:Body>
      ...
   </soap:Body>
</soap:Envelope>
```

Table 1: General example of EPR for service MyService.

An EPR is returned by a WS-Addressing compliant service as a response to an invocation of the service. An EPR can hold several fields but important for this specification are the Address and ReferenceParameters fields. The Address tells the client what URL should be used in future communication (it may or may not be the same as the HTTP connection URL). The optional ReferenceParameters field contains an *opaque* reference to the specific service instance at the Address URL (similar to a ticket or identifier). The EPR is used by the client and the service in subsequent message exchanges.

A central WSRF concept is “WS Resources”. It can be used to represent resources such as a shopping cart, hardware such as a printer or a print job created in a printer. WSRF defines the messages sent between client and service and API functions that the client uses to interact with these resources. A WS Resource is represented by a property document. This document represents the state of the resource and is communicated as XML in the SOAP body.

[4] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

In WSRF there are several APIs that can be used:

- WS-Addressing
- WS-Notification
- WS-Policy
- WS-Security.

However, only WS-Addressing is mandatory. The mandatory operations defined there by WSRF must be implemented by all WSRF web-services. For a full list and information which operations are mandatory and which are optional, please see the official WSRF documentation.

A WS Resource consists of a set of properties (called a document), each with a particular name that is unique within the document. Interaction with WS Resource properties from the client (for example to get or set the value of the properties) is performed by using operations from the WS-Resource specification. If a call with such an API function fails, there are several pre-defined error messages (called Faults). These Faults can also be extended to provide service-specific error messages.

Since WSRF is WS-Addressing compliant, any WSRF compliant service must use an EPR to refer to a particular WS Resource.

Below we list the namespaces used in examples in this document.

soap	http://schemas.xmlsoap.org/soap/envelope/
moby	http://www.biomoby.org/moby
mobyws	http://biomoby.org/
wsa	http://www.w3.org/2005/08/addressing
wsrp	http://docs.oasis-open.org/wsrf/rp-2
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsbf	http://docs.oasis-open.org/wsrf/bf-2

3.2. Operations, faults and properties

Asynchronous BioMOBY services must represent a batch-call as one WS Resource and therefore that the way to refer to such a batch-call is by using the EPR. The version of WSRF used is v1.2.

A BioMOBY asynchronous service must implement the following WSRF operations[5]:

- *GetResourceProperty*: This mandatory WSRF operation (from WS-Resource) allows a requestor to retrieve the value of a *single* resource property of a WS-Resource.
- *GetMultipleResourceProperties*: This optional WSRF operation (from WS-Resource) allows a requestor to retrieve the values of *multiple* resource properties of a WS-Resource.
- *Destroy*: This optional WSRF operation (from WS-ResourceLifetime) allows a service requestor to request the immediate destruction of a WS-Resource.

Furthermore, both BioMOBY asynchronous services and BioMOBY asynchronous clients must be able to process the following standard WSRF faults.

[5] For a detailed documentation about each WSRF specification, operation and fault, please see http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

- *ResourceUnknownFault*: Used to indicate that the resource identified in the message is not known to the Web service.
- *ResourceUnavailableFault*: Used to indicate that the Web service is active, but unable to provide access to the resource.
- *InvalidResourcePropertyQNameFault*: Used to indicate that resource property specified in the request message did not correspond to a resource property element of the WS-Resource referred to in the request message.
- *ResourceNotDestroyedFault*: Used to indicate that a WS-Resource was not destroyed for some reason.

The following groups of properties are mandatory for a BioMOBY asynchronous service:

- *Status* properties:
    - This group represents the status of the jobs resulting from a batch-call. The batch-call itself is identified through the EPR, so the name of the property will be used to identify the job. The service must have a property whose QName is "mobyws:status_*queryID*" for each job (input mobyData). For example, if a batch-call contains two jobs with queryIDs q1 and q2, these properties must exist: "mobyws:status_q1" and "mobyws:status_q2". The status will be given using the OMG's LSAE standard[6] schema for Notification Events.
- *Result* properties:
    - This group represents the result from the particular jobs belonging to a batch-call. If a client asks for a result property before a particular job has finished, or the property is ill-formed (i.e. it is not a valid QName), a WSRF Fault "InvalidResourcePropertyQNameFault" will be returned. The batch-call itself is identified through the EPR, so the name of the property will be used to identify the job. The result properties are QNames similar to the status ones, but following the pattern "mobyws:result_*queryID*".

Please note, that these properties must be read-only. WSRF specifies some optional operations that can be used to delete or modify such properties. The easiest way to not allow access to these operations is to simply not implement them (resulting in a SOAP client fault if a client tries to call them). However, if a service author wishes to implement these operations for modification of other properties of a batch-job not specified here, any attempt to change the properties discussed in this specification must be stopped on the implementation level and an additional WSRF Fault should be returned by the service ("UnableToModifyResourcePropertyFault") signaling that these properties are read-only.

[6] Life Sciences Analysis Engine (LSAE) final adopted specification - http://www.omg.org/cgi-bin/doc?dtc/2005-04-01 BioMOBY clients must be able to understand the pre-defined events in this specification: Heartbeat event, Percent progress event, Job State changed event, Step progress event, Time progress event. For more information, please check the LSAE specification.

3.3. Registering asynchronous services

Services must be able to indicate to clients if their services can work in an asynchronous mode or not. Such information must be made part of the service registration procedure. The category field of a service instance will be used for storing this information.

For reasons of compatibility with synchronous clients, it is mandatory to provide a synchronous version of the service. As before, only the synchronous service will be registered in MobyCentral.

- If a service is provided in synchronous mode only (category is `moby`), the service provider will implement just one SOAP operation that is named exactly like the service (e.g. doBlastAnalysis).
- If a service is provided in asynchronous mode (category is `moby-async`), the service provider must implement the following SOAP operations (in addition to the synchronous operation used for synchronous mode):
    - For submitting (starting) a job:
        - The service name registered in MobyCentral with _submit appended (e.g. doBlastAnalysis_submit). This method will be **RPC/encoded**.
    - For polling the status of a job:
        - *GetResourceProperty/GetMultipleResourceProperties* for the property "status_*qname*", where qname is the queryID of the corresponding mobyData input. This method will be **document/literal**.
    - For retrieving results of a job/jobs:
        - *GetResourceProperty/GetMultipleResourceProperties* for the property/properties "result_*qname*", where qname is the query name of the corresponding mobyData input. This method will be **document/literal**.
    - For destroying a job:
        - *Destroy* for the batch-call identifier created during submission. This method will be **document/literal**.

Please note that although the service provider must implement and publish these additional operations, *only* the synchronous version will be registered in MobyCentral. If asynchronous equals true during service registration, MobyCentral (the registry) will produce WSDL for the additional SOAP methods (and, of course, the service must implement these SOAP methods). Therefore, the knowledge if a service can be called in an asynchronous way must be stored in the registry.

Even if a service is asynchronous, it must always be possible[7] to call it in a synchronous mode. Naturally, in this case it is possible (as before) that the connection between client and service is closed because of a timeout.

3.3. Determining if a service is asynchronous capable

A client discovers services by using the *findService*[8] API call of BioMOBY. We suggest allowing a new value in the input field protocol and the output field category 'moby-async', adding to the normal values 'moby', 'cgi' and 'soap'.

In BioMOBY, clients receive a WSDL from MobyCentral by using the retreiveService operation from the BioMOBY API. Clearly, the operations relating to WSRF must also be described in the WSDL[9] that MobyCentral generates.

---

[7] Note that the SOAP method should be available. If, however, the author of the service knows that the service will *never* finish before a time-out, it is reasonable to return an empty result directly together with a MobyException with error code 701 (Specific errors from the BioMOBY service) and the message "Service must be invoked asynchronously.".

[8] First, we note that the fields *protocol* and *category* fields are named differently but represent the same thing in the input and output. It would be better to keep a consistent naming. Second, the fact that a service is capable of asynchronous communication is *by itself* not a reason to choose this service over another synchronous-only service.

The ability to call a service asynchronously can also be deduced from LSID resolution[10]. We suggest that this information should be stored in the "dc:format" predicate of the RDF describing a service instance. Currently, only the value 'BioMoby_service" is allowed. We suggest adding another allowed value to this predicate to indicate that it is a BioMOBY service and that it can be called asynchronously (remember that asynchronous also implies that there is a synchronous way to call the service). Because values for this predicate is related to the myGrid ontology, we suggest to use an existing and appropriate value from this ontology or create a new value in collaboration with the designers of this ontology.

3.3. Invoking asynchronous services – Communication sequence

The sequence for an asynchronous MOBY service execution is the following:

A. The client starts the session
   - If the client is only capable of synchronous communication, the services (both sync and async) will work in synchronous mode (current BioMOBY behavior)
   - If the client and service are capable of asynchronous communication, the client sends the request message to the ***servicename*_submit** SOAP operation to inform the service that they wish to run the service in asynchronous mode, and that it is able to cope with asynchronous calls. The message contains the standard MOBY content and data needed to execute the service.

   If the service is synchronous, then it will construct the result and return it. However, asynchronous services will accept the task and return back a WSRF message (containing a batch-call identifier or "ticket" inside of an EPR), instead of the actual result. From this point both client and server will work in asynchronous modes.

B. Polling for service execution status

   The client uses the **GetResourceProperty/GetMultipleResourceProperties** WSRF operation to retrieve status information using the batch-call identifier (in the EPR) and the queryID of the corresponding mobyData input. This step can be repeated as many times as needed until the service execution is done and the result is ready, using the SOAP operation to periodically request for execution status information. Additional notification information (i.e. "step 2: sequence alignment done") can be included to report progress status from the requested service to the client.

C. Request service execution result

   When the client receives a status from the service that indicates that the execution is completed, the client can ask the service for the result by sending a message containing the asynchronous batch-call identifier and queryID to the **GetResourceProperty/ GetMultipleResourceProperties** WSRF operation for the properties in the "result" group.

   The response from this operation call will contain WS Resource properties whose formats are MOBY compatible and will include, any MobyExceptions that might have occurred[11]. These properties are all available until the resource is no longer available. A resource becomes unavailable when the service either by its own initiative cleans the result or after a client explicitly asks that the resource is cleaned.

D. The client destroys the session

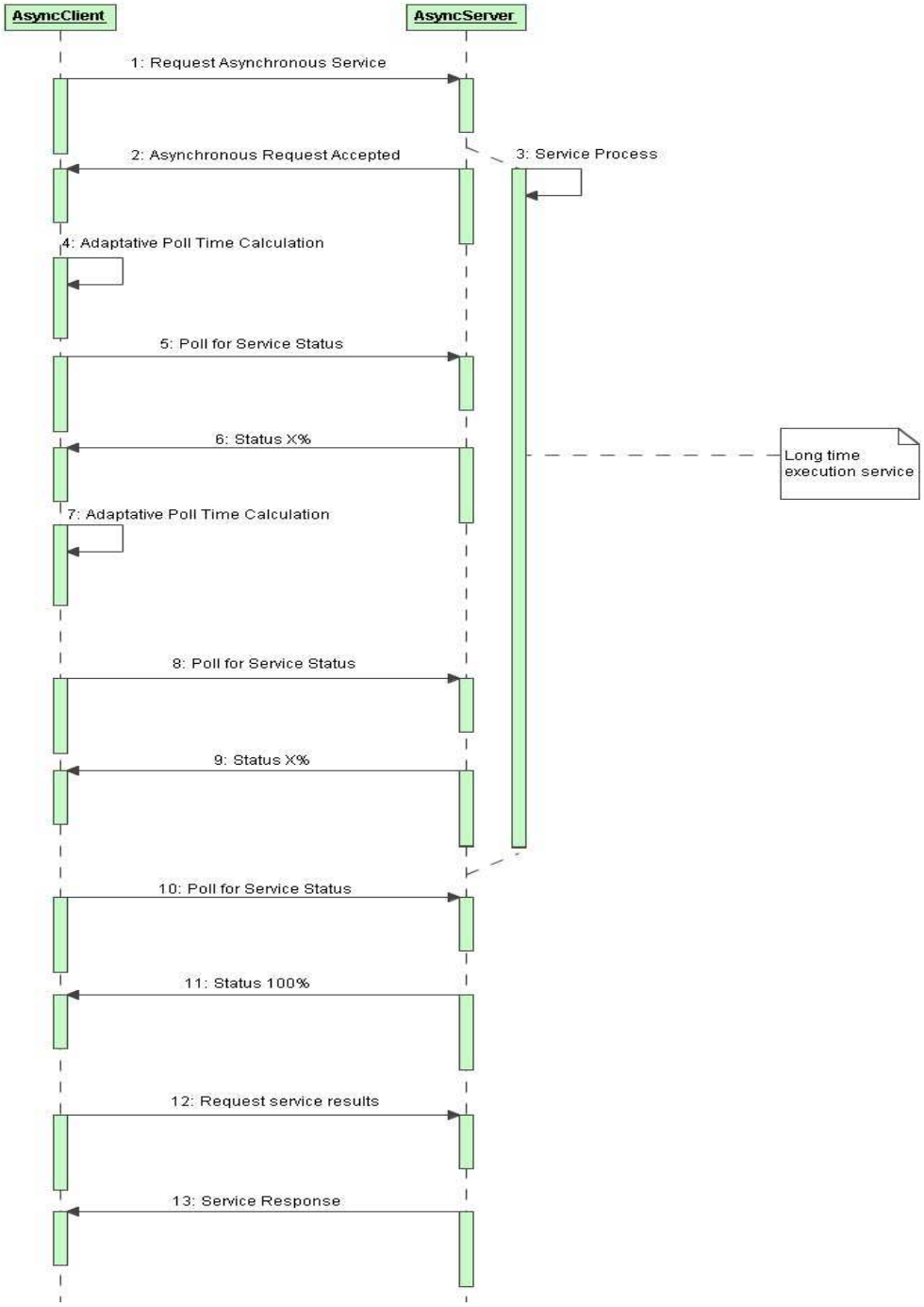
---

[9] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[10] http://biomoby.open-bio.org/index.php/for-developers/moby_extensions/moby_metadata

[11] Note that this means that if an error relating to Moby happens (for example, not properly formatted BioMOBY input or invalid object), then the proper way to handle this situation is the following: report a *status* using LSAE that signals that the job is finished (several options in LSAE for this) and then report a *result* as a normal Moby message with whatever MobyException that is suitable.

Once the client has retrieved the results of all jobs and do not wish to retrieve them again, it should destroy the session created during the step A. This is done by sending a message containing the EPR to the **Destroy** WSRF operation[12].



Sequence diagram for an async-async communication using BioMOBY

[12] It is possible that the service provider by his own initiative cleans the results of old batch-calls. This depends however on the policy of the service provider and no "minimum" time that the results should be available is specified but clearly most providers might want to clean "old" results at some point. If the resource has been cleaned and a client request the resource then the service should respond with the WSRF Fault "ResourceUnknownFault" (since there no longer exists any trace of this resource/batch-call)

3.4. Asynchronous messages

Service execution request is the only message that will follow exactly the normal BioMOBY standard, the other messages will be WSRF messages.

Regarding error-handling: naturally, in all these SOAP calls it is possible that we get SOAP related faults but we do not specify these here.

When using the standard WSRF operations it is possible to get the standard WSRF faults. We give a general example in the appendix to show readers how such a fault might look like. In this section we list the documented WSRF faults in each situation. For more information about WSRF operations and faults, we again refer the reader to the official WSRF documentation.

The messages are as follows:

1. **Requesting asynchronous service execution:** This message is identical to the BioMOBY XML to request synchronous service execution. The only difference is that the client sends the request to the ***servicename*_submit** SOAP method.

SOAP XML request for asynchronous service execution (updated)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      ...
   </soap:Header>
   <soap:Body wsu:Id="myBody">
      <servicename_submit xmlns="http://biomoby.org/">
         <c-gensym9 xsi:type="xsd:string">
            &lt;?xml version='1.0' encoding='UTF-8'?&gt;
            &lt;moby:MOBY xmlns:moby='http://www.biomoby.org/moby'&gt;
               &lt;moby:mobyContent&gt;
                  &lt;moby:mobyData queryID='queryId00'&gt;
                     &lt;!--- Standard BioMOBY XML for Input --&gt;
                  &lt;/moby:mobyData&gt;
                  &lt;moby:mobyData queryID='queryId01'&gt;
                     &lt;!--- Standard BioMOBY XML for Input --&gt;
                  &lt;/moby:mobyData&gt;
                  ...
               &lt;/moby:mobyContent&gt;
            &lt;/moby:MOBY&gt;
         </c-gensym9>
      </servicename_submit>
   </soap:Body>
</soap:Envelope>
```

2. **Accepted asynchronous request:** The service recognizes the asynchronous request, and communicates to the client that its request was accepted and that the service will work in asynchronous mode. For this accepted asynchronous request (and for the polling, and polling response operations), a standard WSA compliant message is sent, containing two important parts in the EPR:

   - **`Address`**: This is the address where the EPR is a valid reference to the resulting batch-call. Following the recommendation from WS-Resource 1.2, this address is a composition of the original end-point plus the resource identified by the job ticket.
   - **`ReferenceParameters`**: The ticket representing the service provider identifier for the service execution job. The value of this ticket has no intrinsic meaning. The service provider can choose it be any legal XML fragment. Clients should *not* attempt to interpret the value of the ticket; it is simply an identifier and should remain opaque from the point of view of the client.

SOAP XML response for accepted asynchronous service execution (New)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      ...
   </soap:Header>
   <soap:Body>
      <servicename_submitResponse xmlns="http://biomoby.org/">
       <namesp1:body xmlns:namesp1="http://biomoby.org/">
         <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:Address>http://myserver.com/MyService?asyncId=ID</wsa:Address>
            <wsa:ReferenceParameters>
               <mobyws:ServiceInvocationId xmlns:mobyws="http://biomoby.org/">
                  ID
               </mobyws:ServiceInvocationId>
            </wsa:ReferenceParameters>
         </wsa:EndpointReference>
       </namesp1:body>
      </servicename_submitResponse>
   </soap:Body>
</soap:Envelope>
```

This EndPointReference is included in the SOAP header in the subsequent messages and is used to refer to the batch-call.

3. **Polling for service status:** We assume a polling model where the client queries the service asking for the status of its request. The client makes the polling requests by sending a message to the GetResourceProperty or GetMultipleResourceProperties WSRF operations of the service. The structure of the polling message is an element labeled as a WSA reference parameter with the asynchronous job "ticket" (in the SOAP header) and the original request query identifiers encoded as QNames associated to mobyws namespace (properties mobyws:`status_`*`queryID`*) in the WSRF message within SOAP body.

SOAP XML request to poll for service execution status: (New)

GetResourceProperty

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetResourceProperty/GetResourcePropertyRequest
      </wsa:Action>
      <wsa:To wsu:Id="To">http://myserver.com/MyService</wsa:To>
      <mobyws:ServiceInvocationId wsa:IsReferenceParameter="true">
         ID
      </mobyws:ServiceInvocationId>
      ...
   </soap:Header>
   <soap:Body>
      <GetResourceProperty xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         mobyws:status_queryId00
      </GetResourceProperty>
   </soap:Body>
</soap:Envelope>
```

SOAP XML request to poll for service execution status: (New)

GetMultipleResourceProperties

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest
      </wsa:Action>
      <wsa:To wsu:Id="To">http://myserver.com/MyService</wsa:To>
      <mobyws:ServiceInvocationId wsa:IsReferenceParameter="true">
         ID
      </mobyws:ServiceInvocationId>
      ...
   </soap:Header>
   <soap:Body>
      <wsrp:GetMultipleResourceProperties
               xmlns:wsrp="http://docs.oasis-open.org/wsrf/rp-2">
         <wsrp:ResourceProperty>
               mobyws:status_queryId00
         </wsrp:ResourceProperty>
         <wsrp:ResourceProperty>
               mobyws:status_queryId01
         </wsrp:ResourceProperty>
         ...
      </wsrp:GetMultipleResourceProperties>
   </soap:Body>
</soap:Envelope>
```

Valid fault messages are ResourceUnknownFault, ResourceUnavailableFault and InvalidResourcePropertyQNameFault.

4. **Response for polling request:** The response sent from the service includes the current process status for the requested query identifiers.

SOAP XML response for polling request: (New)

GetResourceProperty

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:From wsu:Id="From">
         <wsa:Address>http://myserver.com/MyService</wsa:Address>
      </wsa:From>
       <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
          GetResourceProperty/GetResourcePropertyResponse
       </wsa:Action>
      ...
   </soap:Header>
   <soap:Body>
      <GetResourcePropertyResponse
               xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         <mobyws:status_queryId00>
            <!-- LSAE Analysis Event Block -->
         </mobyws:status_queryId00>
      </GetResourcePropertyResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP XML response for polling request: (New)

GetMultipleResourceProperties

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:From wsu:Id="From">
         <wsa:Address>http://myserver.com/MyService</wsa:Address>
      </wsa:From>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetMultipleResourceProperties/GetMultipleResourcePropertiesResponse
      </wsa:Action>
      ...
   </soap:Header>
   <soap:Body>
      <GetMultipleResourcePropertiesResponse
               xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         <mobyws:status_queryId00>
            <!-- LSAE Analysis Event Block -->
         </mobyws:status_queryId00>
         <mobyws:status_queryId01>
            <!-- LSAE Analysis Event Block -->
         </mobyws:status_queryId01>
         ...
      </GetMultipleResourcePropertiesResponse>
   </soap:Body>
</soap:Envelope>
```

Valid fault messages are ResourceUnknownFault, ResourceUnavailableFault and InvalidResourcePropertyQNameFault.

5. **Requesting the result:** Once the client knows that the service execution is finished and that the result is ready, it should retrieve results by sending a message to the GetResourceProperty or GetMultipleResourceProperties WSRF operations in the server. The message structure is the same as in the step before, but in this case asking for mobyws:`result_`*`queryID`* properties (encoded as QNames associated to mobyws namespace).

SOAP XML request for the result: (New)

GetResourceProperty

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetResourceProperty/GetResourcePropertyRequest
      </wsa:Action>
      <wsa:To wsu:Id="To">http://myserver.com/MyService</wsa:To>
      <mobyws:ServiceInvocationId wsa:IsReferenceParameter="true">
         ID
      </mobyws:ServiceInvocationId>
      ...
   </soap:Header>
   <soap:Body>
      <GetResourceProperty xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         mobyws:result_queryId00
      </GetResourceProperty>
   </soap:Body>
</soap:Envelope>
```

SOAP XML request for the result: (New)

GetMultipleResourceProperties

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest
       </wsa:Action>
      <wsa:To wsu:Id="To">http://myserver.com/MyService</wsa:To>
      <mobyws:ServiceInvocationId wsa:IsReferenceParameter="true">
         ID
      </mobyws:ServiceInvocationId>
      ...
   </soap:Header>
   <soap:Body>
      <wsrp:GetMultipleResourceProperties
               xmlns:wsrp="http://docs.oasis-open.org/wsrf/rp-2">
         <wsrp:ResourceProperty>
               mobyws:result_queryId00
         </wsrp:ResourceProperty>
         <wsrp:ResourceProperty>
               mobyws:result_queryId01
         </wsrp:ResourceProperty>
         ...
      </wsrp:GetMultipleResourceProperties>
   </soap:Body>
</soap:Envelope>
```

6. **Response for requesting the result:** The content of the requested mobyws:`result_`*`queryID`* properties is a standard BioMOBY response message containing the result of the service execution. Note that if a BioMOBY related error occured during the execution, the BioMOBY response will contain empty *mobyData* with corresponding *mobyException*.

SOAP XML response for requesting results: (New)

GetResourceProperty

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:From wsu:Id="From">
          <wsa:Address>http://myserver.com/MyService</wsa:Address>
      </wsa:From>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetResourceProperty/GetResourcePropertyResponse
      </wsa:Action>
      ...
   </soap:Header>
   <soap:Body>
    <GetResourcePropertyResponse xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         <mobyws:result_queryId00>
            <moby:MOBY xmlns:moby='http://www.biomoby.org/moby'>
               <moby:mobyContent>
                  <moby:mobyData queryID='queryId00'>
                     <!--- Standard BioMOBY XML for output -->
                  </moby:mobyData>
               </moby:mobyContent>
            </moby:MOBY>
         </mobyws:result_queryId00>
    </GetResourcePropertyResponse>
   </soap:Body>
</soap:Envelope>
```

SOAP XML response for requesting results: (New)

GetMultipleResourceProperties

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:From wsu:Id="From">
         <wsa:Address>http://myserver.com/MyService</wsa:Address>
      </wsa:From>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/
         GetMultipleResourceProperties/GetMultipleResourcePropertiesResponse
      </wsa:Action>
      ...
   </soap:Header>
   <soap:Body>
      <GetMultipleResourcePropertiesResponse
                     xmlns="http://docs.oasis-open.org/wsrf/rp-2">
         <mobyws:result_queryId00>
            <moby:MOBY xmlns:moby='http://www.biomoby.org/moby'>
               <moby:mobyContent>
                  <moby:mobyData queryID='queryId00'>
                     <!--- Standard BioMOBY XML for output -->
                  </moby:mobyData>
               </moby:mobyContent>
            </moby:MOBY>
         </mobyws:result_queryId00>
         <mobyws:result_queryId01>
            <moby:MOBY xmlns:moby='http://www.biomoby.org/moby'>
               <moby:mobyContent>
                  <moby:mobyData queryID='queryId01'>
                     <!--- Standard BioMOBY XML for output -->
                  </moby:mobyData>
               </moby:mobyContent>
            </moby:MOBY>
         </mobyws:result_queryId01>
         ...
      </GetMultipleResourcePropertiesResponse>
   </soap:Body>
</soap:Envelope>
```

It will be a client task (at API level) to compose a standard BioMOBY message from the WSRF response.
Valid fault messages are ResourceUnknownFault, ResourceUnavailableFault and InvalidResourcePropertyQNameFault.

7. **Destroying the resource for asynchronous service execution**: After client has retrieved the results of all query identifiers, it should destroy the resource it was created during asynchronous service execution request by sending a message to the Destroy WSRF operation in the server. The structure of this message is an element labeled as a WSA reference parameter with the asynchronous job ticket (in the SOAP header).

SOAP XML request for destroying the resource (New)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:Action>http://docs.oasis-open.org/wsrf/rlw-2/
         ImmediateResourceTermination/DestroyRequest
      </wsa:Action>
      <wsa:To wsu:Id="To">http://myserver.com/MyService</wsa:To>
      <mobyws:ServiceInvocationId wsa:IsReferenceParameter="true">
         ID
      </mobyws:ServiceInvocationId>
      ...
   </soap:Header>
   <soap:Body>
      <Destroy xmlns="http://docs.oasis-open.org/wsrf/rl-2"/>
   </soap:Body>
</soap:Envelope>
```

8. **Response for destroying the resource**:

SOAP XML response for destroying the resource (New)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:wsa="http://www.w3.org/2005/08/addressing"
               xmlns:mobyws="http://biomoby.org/">
   <soap:Header>
      ...
      <wsa:From wsu:Id="From">
         <wsa:Address>http://myserver.com/MyService</wsa:Address>
      </wsa:From>
      <wsa:Action>http://docs.oasis-open.org/wsrf/rlw-2/
         ImmediateResourceTermination/DestroyResponse
      </wsa:Action>
      ...
   </soap:Header>
   <soap:Body>
      <DestroyResponse xmlns="http://docs.oasis-open.org/wsrf/rl-2"/>
   </soap:Body>
</soap:Envelope>
```

Accepted fault messages are ResourceUnknownFault, ResourceUnavailableFault and ResourceNotDestroyedFault.

APPENDIX A – WSRF Faults

In this appendix we give brief example of a WSRF Fault:

WSRF fault (New)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                    xmlns:wsa="http://www.w3.org/2005/08/addressing"
                    xmlns:mobyws="http://biomoby.org/">
  <soap:Header>
    ...
    <wsa:From wsu:Id="From">
      <wsa:Address>http://myserver.com/MyService</wsa:Address>
    </wsa:From>
    <wsa:Action wsu:Id="Action">
       http://docs.oasis-open.org/wsrf/fault
    </wsa:Action>
    ...
  </soap:Header>
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Application error</faultstring>
      <detail>
        <wsbf:FaultMessage>
          <wsbf:Timestamp>fault_timestamp</wsbf:Timestamp>
          <wsbf:Description>fault_description</wsbf:Description>
        </wsbf:FaultMessage>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

These errors come as standard SOAP Faults but with WSRF specific faults inside the details section. All WSRF faults are extended from the standard WSRF BaseFault using the XML Schema extension mechanism. For details, see WS-BaseFaults[13].

[13] http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf

APPENDIX B – Errata

This section contains the changes of this document, with the baseline at v2.3:

Version 2.4:

- Explicitly states that WSRF v1.2 is supported.
- Submit method uses RPC/encoded for both input and output.
- Explicit declaration for each operation (WSRF and MOBY) about which encoding they use.

  The WSDL template has been remade for asynchronous services. In the version 2.3, it had a mix of RPC/encoded and document/literal in the only binding it had, and we have inferred from WSDL specs that is not allowed in WSDL 1.1.

  The solution has been distributing the operations on two disjunctive port types and bindings, one for RPC/encoded operations and the other for document/literal operations. In that way, the service has two ports, one for MOBY and the other for WSRF, both of them using the same endpoint.
- Added namespaces to the examples.

  In this version, the namespaces have been specified for the communication operations (SOAP, WSRF, and MOBY).
- Appendix C with short tutorial (and overview) for the Perl library for developing asynchronous services.

Version 2.4.1:

- Sample SOAP message under *"Requesting asynchronous service execution"* was wrong, so it has been updated. (*Thanks to Sergio Ramírez*)
- Sample WSDL and Perl codes in Appendix C have been highlighted.
- Sample WSDL was encoded in UTF-8.

Version 2.4.2:

- *MOBY::Async* client and service libraries, and sample SOAP messages were not WSRF compliant, so they have been updated. (*Thanks to Dmitry Repchevsky*)
- Sample WSDL and Perl codes in Appendix C have been rewritten using as source results derived from new release of *MOBY::Async*.
- Nomenclature and explanations in some paragraphs related to messages flow is now more correct.

APPENDIX C – Constructing Asynchronous MOBY Compliant Services

Asynchronous MOBY Services follows the same paradigm as traditional synchronous MOBY services: they have a single SOAP server running as a CGI script (listener) and this listener hands-off requests to the appropriate code module as requests arrive.

The dispatcher

Below you can find an example of source code for a typical SOAP server CGI adapted for asynchronous MOBY services. The *MOBY::Async::WSRF* module hides all the SOAP and WSRF complexity. It is based in the *WSRF::Lite* Perl implementation of the WSRF Standard, which is based in the *SOAP::Lite* Perl implementation of the SOAP Standard.

```
#!/usr/bin/perl -w
use SOAP::Transport::HTTP;
use MOBY::Async::WSRF;
use lib '/usr/local/apache2/cgi-bin/MOBY/'; # the location of your Service modules
use Services::AsyncServices; # the name of your Service modules

my $server = new SOAP::Transport::HTTP::CGI;
$server->serializer(WSRF::Serializer->new);
$server->deserializer(WSRF::Deserializer->new);

# Magic! It works!
$server->on_action(sub{});

$server->dispatch_with({
        $WSRF::Constants::MOBY.'#sayHello'        => 'Services::AsyncServices',
        $WSRF::Constants::MOBY.'#sayHello_submit' => 'Services::AsyncServices',
        $WSRF::Constants::WSRPW.'/GetResourceProperty/GetResourcePropertyRequest' =>
                'Services::AsyncServices',
        $WSRF::Constants::WSRPW.'/GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest' =>
                'Services::AsyncServices',
        $WSRF::Constants::WSRLW.'/ImmediateResourceTermination/DestroyRequest' =>
                'Services::AsyncServices',
});
$server->handle();
```

Here, an instance of a SOAP CGI server is created (listener), selecting both WSRF Serializer and WSRF Deserializer for the SOAP CGI server. Then, it makes a mapping between SOAP Action headers and code modules to call some methods.

Both the method “sayHello” (synchronous service requests) as the method “sayHello_submit” (asynchronous service requests) will be passed to the Services::AsyncServices module.

The methods defined in the WS-ResourceProperties spec. ("GetResourceProperty" and "GetMultipleResourceProperties") and the methods of WS-ResourceLifetime spec. ("Destroy") will be passed to the same module, as well.

The service module

The dispatcher merely listens for incoming MOBY service requests, and then passes them off to the appropriate Perl module (in this case *Services::AsyncServices*) where the real work gets done.

Now let's look at the structure of the *Services::AsyncServices* module. Generally speaking, it is just a regular Perl module, but which extends the *MOBY::Async::SimpleServer* base class. It has the following structure:

```
package Services::AsyncServices;
use strict;
use SOAP::Lite;
use MOBY::CommonSubs qw(:all);
use MOBY::Async::SimpleServer;
use base qw(MOBY::Async::SimpleServer);
```

We are using *MOBY::CommonSubs* because it exports many useful subroutines that we will require to handle incoming MOBY objects and output valid MOBY responses. Services must take incoming data, parse it into individual (enumerated) requests, process each request, generate a MOBY-compliant response message for each request, and then return the (identically enumerated) response messages, along with optional server-side provision information.

By extending the *MOBY::Async::SimpleServer* base class, the *Services::AsyncServices* module inherits all its methods: WS-ResourceProperties methods (GetResourceProperties and GetMultipleResourceProperties) and WS-ResourceLifetime methods (Destroy).

It is also necessary to define the methods which attend to the synchronous and asynchronous requests of the service (“sayHello” and “sayHello_submit” respectively in the dispatcher example). For this purpose, *MOBY::Async::SimpleServer* provides sync, error and async methods that can be used to get the desired behaviour.

```
package Services::AsyncServices;
use strict;
use SOAP::Lite;
use MOBY::CommonSubs qw(:all);
use MOBY::Async::SimpleServer;
use base qw(MOBY::Async::SimpleServer);

# This variable is a subroutine which carry out the core of the service
my $sayHello = sub {
        # ...
        # The code of your service
        # ...
};

# This is the method that answers to synchronous requests
sub sayHello {
        my $self = shift @_;
        # Here you can choose between sync or error
        #return $self->sync($sayHello, $TIMEOUT, @_);
        return $self->error(@_);
}

# This is the method that answers to asynchronous requests
sub sayHello_submit {
        my $self = shift @_;
        return $self->async($sayHello, @_);
}

1;
```

For attending synchronous requests (“sayHello” subroutine) you can choose between:

- Directly return a moby exception indicating that the service must be invoked asynchronously:

```
return $self->error(@_);
```

- Try to execute the service and return a MOBY exception if it exceed a defined time out:

```
return $self->sync($sayHello, $TIMEOUT, @_);
```

  sync expects three parameters: a variable which is the subroutine that will execute the service (“$sayHello”); the allowed timeout for executing the service (“$TIMEOUT”); and the input data of the service (“@_”).

For attending asynchronous request (“sayHello” subroutine) you should use *async* method. It basically forks the main process as many times as mobyData are contained into the input message, and returns, leaving the child processes running in background:

```
return $self->async($sayHello, @_);
```

async expects two parameters: a variable which is the subroutine that will execute the service (“$sayHello”); and the input data of the service (“@_”).

The WSDL

Next, we are going to show the WSDL that is generated by MOBYCentral for this BioMOBY asynchronous service.

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

<wsdl:definitions name="MOBY_Central_Generated_WSDL"
   targetNamespace="http://biomoby.org/Central.wsdl"
   xmlns:tns="http://biomoby.org/Central.wsdl"
   xmlns:xsd1="http://biomoby.org/CentralXSDs.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsoap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsrp="http://docs.oasis-open.org/wsrf/rp-2"
   xmlns:wsrl="http://docs.oasis-open.org/wsrf/rl-2"
   xmlns:wsbf="http://docs.oasis-open.org/wsrf/bf-2"
   xmlns:wsrpw="http://docs.oasis-open.org/wsrf/rpw-2"
   xmlns:wsrlw="http://docs.oasis-open.org/wsrf/rlw-2"
   xmlns:wsrw="http://docs.oasis-open.org/wsrf/rw-2"
   xmlns:wsa="http://www.w3.org/2005/08/addressing">

   <wsdl:import
      namespace="http://docs.oasis-open.org/wsrf/rpw-2"
      location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl"/>
   <wsdl:import
      namespace="http://docs.oasis-open.org/wsrf/rlw-2"
      location="http://docs.oasis-open.org/wsrf/rlw-2.wsdl"/>
   <wsdl:import
      namespace="http://docs.oasis-open.org/wsrf/rw-2"
      location="http://docs.oasis-open.org/wsrf/rw-2.wsdl"/>

   <wsdl:types>
      <xsd:schema elementFormDefault="qualified"
         targetNamespace="http://biomoby.org/Central.wsdl"
         xmlns="http://biomoby.org/Central.wsdl"
      >
         <xsd:import
            namespace="http://docs.oasis-open.org/wsrf/bf-2"
            schemaLocation="http://docs.oasis-open.org/wsrf/bf-2.xsd"/>
         <xsd:import
            namespace="http://docs.oasis-open.org/wsrf/rl-2"
            schemaLocation="http://docs.oasis-open.org/wsrf/rl-2.xsd"/>
         <xsd:import
            namespace="http://www.w3.org/2005/08/addressing"
            schemaLocation="http://www.w3.org/2002/ws/addr/ns/ws-addr" />
         <xsd:complexType name="MOBY_async_OutputType">
            <xsd:sequence minOccurs="1" maxOccurs="1">
               <xsd:element ref="wsa:EndpointReference"/>
            </xsd:sequence>
         </xsd:complexType>
```

```
      <xsd:element name="ResourceProperties">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:any minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</wsdl:types>

<wsdl:message name="sayHelloInput">
   <wsdl:part name="data" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="sayHelloOutput">
   <wsdl:part name="body" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="sayHello_submitInput">
   <wsdl:part name="data" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="sayHello_submitOutput">
   <wsdl:part name="body" type="tns:MOBY_async_OutputType"/>
</wsdl:message>

<wsdl:portType name="sayHelloPortType"
   wsrp:ResourceProperties="tns:ResourceProperties">
   <wsdl:operation name="sayHello">
      <wsdl:input message="tns:sayHelloInput"/>
      <wsdl:output message="tns:sayHelloOutput"/>
   </wsdl:operation>
   <wsdl:operation name="sayHello_submit">
      <wsdl:input message="tns:sayHello_submitInput"/>
      <wsdl:output message="tns:sayHello_submitOutput"/>
   </wsdl:operation>
</wsdl:portType>

<wsdl:portType name="WSRF_Operations_PortType"
   wsrp:ResourceProperties="tns:ResourceProperties">
   <wsdl:operation name="GetResourceProperty">
      <wsdl:input
         name="GetResourcePropertyRequest"
         message="wsrpw:GetResourcePropertyRequest"/>
      <wsdl:output
         name="GetResourcePropertyResponse"
         message="wsrpw:GetResourcePropertyResponse"/>
      <wsdl:fault name="ResourceUnknownFault" message="wsrw:ResourceUnknownFault"/>
      <wsdl:fault
         name="ResourceUnavailableFault"
         message="wsrw:ResourceUnavailableFault"/>
      <wsdl:fault name="InvalidResourcePropertyQNameFault"
         message="wsrpw:InvalidResourcePropertyQNameFault"/>
   </wsdl:operation>
   <wsdl:operation name="GetMultipleResourceProperties">
      <wsdl:input name="GetMultipleResourcePropertiesRequest"
         message="wsrpw:GetMultipleResourcePropertiesRequest"/>
      <wsdl:output name="GetMultipleResourcePropertiesResponse"
         message="wsrpw:GetMultipleResourcePropertiesResponse"/>
      <wsdl:fault name="ResourceUnknownFault" message="wsrw:ResourceUnknownFault"/>
      <wsdl:fault
         name="ResourceUnavailableFault"
         message="wsrw:ResourceUnavailableFault"/>
      <wsdl:fault name="InvalidResourcePropertyQNameFault"
         message="wsrpw:InvalidResourcePropertyQNameFault"/>
   </wsdl:operation>
   <wsdl:operation name="Destroy">
      <wsdl:input message="wsrlw:DestroyRequest"/>
      <wsdl:output message="wsrlw:DestroyResponse"/>
      <wsdl:fault name="ResourceUnknownFault" message="wsrw:ResourceUnknownFault"/>
      <wsdl:fault
         name="ResourceUnavailableFault"
         message="wsrw:ResourceUnavailableFault"/>
      <wsdl:fault
         name="ResourceNotDestroyedFault"
         message="wsrlw:ResourceNotDestroyedFault"/>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="sayHelloBinding" type="tns:sayHelloPortType">
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="sayHello">
      <soap:operation soapAction="http://biomoby.org/#sayHello" style="rpc"/>
      <wsdl:input>
         <soap:body use="encoded" namespace="http://biomoby.org/"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="encoded" namespace="http://biomoby.org/"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
            </wsdl:output>
         </wsdl:operation>
         <wsdl:operation name="sayHello_submit">
            <soap:operation soapAction="http://biomoby.org/#sayHello_submit" style="rpc"/>
            <wsdl:input>
               <soap:body use="encoded" namespace="http://biomoby.org/"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="encoded" namespace="http://biomoby.org/"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
         </wsdl:operation>
      </wsdl:binding>
      <wsdl:binding name="WSRF_Operations_Binding" type="tns:WSRF_Operations_PortType">
         <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
         <wsdl:operation name="GetResourceProperty">
            <soap:operation
              soapAction="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourceProperty/GetResourcePropertyRequest"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
               <soap:fault name="ResourceUnknownFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
               <soap:fault name="ResourceUnavailableFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="InvalidResourcePropertyQNameFault">
               <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="GetMultipleResourceProperties">
            <soap:operation
              soapAction="http://docs.oasis-open.org/wsrf/rpw-
2/GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
               <soap:fault name="ResourceUnknownFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
               <soap:fault name="ResourceUnavailableFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="InvalidResourcePropertyQNameFault">
               <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Destroy">
            <soap:operation
              soapAction="http://docs.oasis-open.org/wsrf/rlw-
2/ImmediateResourceTermination/DestroyRequest"/>
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
               <soap:fault name="ResourceUnknownFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
               <soap:fault name="ResourceUnavailableFault" use="literal"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceNotDestroyedFault">
               <soap:fault name="ResourceNotDestroyedFault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="sayHelloService">
         <wsdl:documentation>Authority: cnio.es  -  MOBY services that check the Async MOBY
libraries</wsdl:documentation>
         <wsdl:port name="sayHelloPort" binding="tns:sayHelloBinding">
            <soap:address location="http://10.10.117.13/cgi-bin/async-MOBY-test.cgi"/>
         </wsdl:port>
         <wsdl:port name="WSRF_Operations_Port" binding="tns:WSRF_Operations_Binding">
            <soap:address location="http://10.10.117.13/cgi-bin/async-MOBY-test.cgi"/>
         </wsdl:port>
      </wsdl:service>
```

```
</wsdl:definitions>
```

Here we could see the dual port types and bindings, one for RPC/encoded operations and document/literal ones. This means that the WSDL two ports, one for MOBY (*sayHelloPort*) and the other for WSRF (*WSRF_Operations_Port)*, both of them using the same SOAP address.