BioMOBY Asynchronous Service Call Proposal

Call Proposal GNV5-06/01

(v 0.6)

Contributions:

Enrique de Andrés
José-María Fernández
Johan Karlsson
Sergio Ramírez
José Manuel Rodríguez Carrasco
Roman Rosset

Coordinators: Oswaldo Trelles, David González-Pisano

Node GNV-5: Integrated Bioinformatics

University of Málaga

Málaga, 26th January 2006

INSTITUTO NACIONAL DE BIOINFORMÁTICA

1. - Preliminaries

This document contains the proposal from INB[1] of how to deal with asynchronous services in BioMOBY. The proposal 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.

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

The main motivation for the proposal 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.

[1] Instituto Nacional de Bioinformática (INB), Spain

2- Current BioMOBY specification

In the version[2] of BioMOBY currently available, 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 (method), by using the name the service was registered with in the MOBY Central catalog.

After retrieving a service description (currently in the form of an illegitimate WSDL document) from MOBY Central, client programs will subsequently communicate directly with the service provider. 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. (NOTE: This becomes the SOAP Action in a SOAP message and the WSDL document describing it; fill in appropriately for your libraries, as it is different for Perl `SOAP::Lite`, `Apache::Axis`, and so on)

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 the current BioMOBY 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.

[2] MOBY-S 0.86.3 API (web-services based)

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

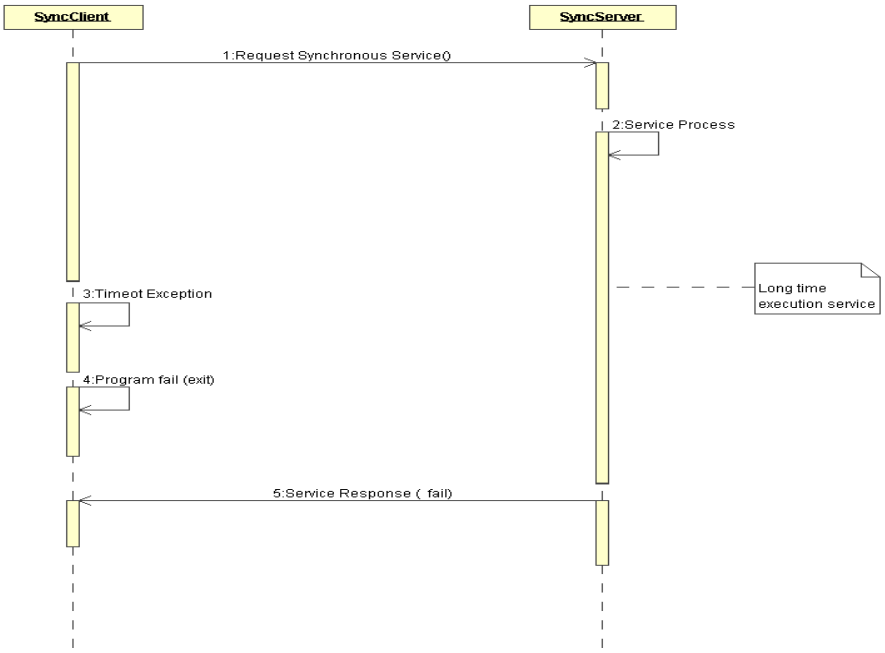


Figure 1 – Illustration of the problem with long-running services in the current BioMOBY specification

3- Specification Proposal

To enable calls to long-running services, it is better to dedicate a separate connection for each step (asynchronous communication model). Unfortunately, the current BioMOBY specification does not support asynchronous calls.

In this chapter, we suggest how to add such support to the BioMOBY specification.

3.1. Registering asynchronous services

Service providers must indicate to clients if their services can work in an asynchronous mode or not. Such metadata information should be made part of the service registration procedure. We propose to add a boolean parameter to the service registration API call (`asynchronous`).

As before, communication between client and service is performed with SOAP RPC calls: the name of the remote procedure call is the same as the name of the service when it was registered in MOBY Central, or the name with a suffix (only for asynchronous services).

- If a service is provided in synchronous mode only (`asynchronous=false`), the service provider will implement just one SOAP method that is named exactly like the service (e.g. doBlastAnalysis).
- If a service is provided in asynchronous mode (`asynchronous=true`), the service provider must implement three SOAP methods (in addition to the synchronous method used for synchronous mode) that are named exactly like the service but with appended suffixes:
    - The name with _async appended (e.g. doBlastAnalysis_async)
    - The name with _poll appended (e.g. doBlastAnalysis_poll)
    - The name with _result appended (e.g. doBlastAnalyis_result)

Even if a service is asynchronous, it must always be possible to call it in a synchronous mode. Naturally, if a client invoke services registered as asynchronous but in a synchronous mode, it is possible (as before) that the connection between client and service can be closed due to timeouts. While a synchronous method is mandatory for every service, asynchronous methods are optional, and they must only be implemented if a service has been registered as asynchronous.

A client that needs to find out if a service is capable of asynchronism must check the service metadata. The proper way to do this has been suggested in RFC 1914: use the LSID of the service to get back metadata as RDF.

To store such information in the RDF metadata we need a new tag, for example:

```
<mobyPred:isAsynchronous rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
       true
</mobyPred:isAsynchronous>
```

3.2. 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*_async** SOAP method 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.

B. The server accepts the client request

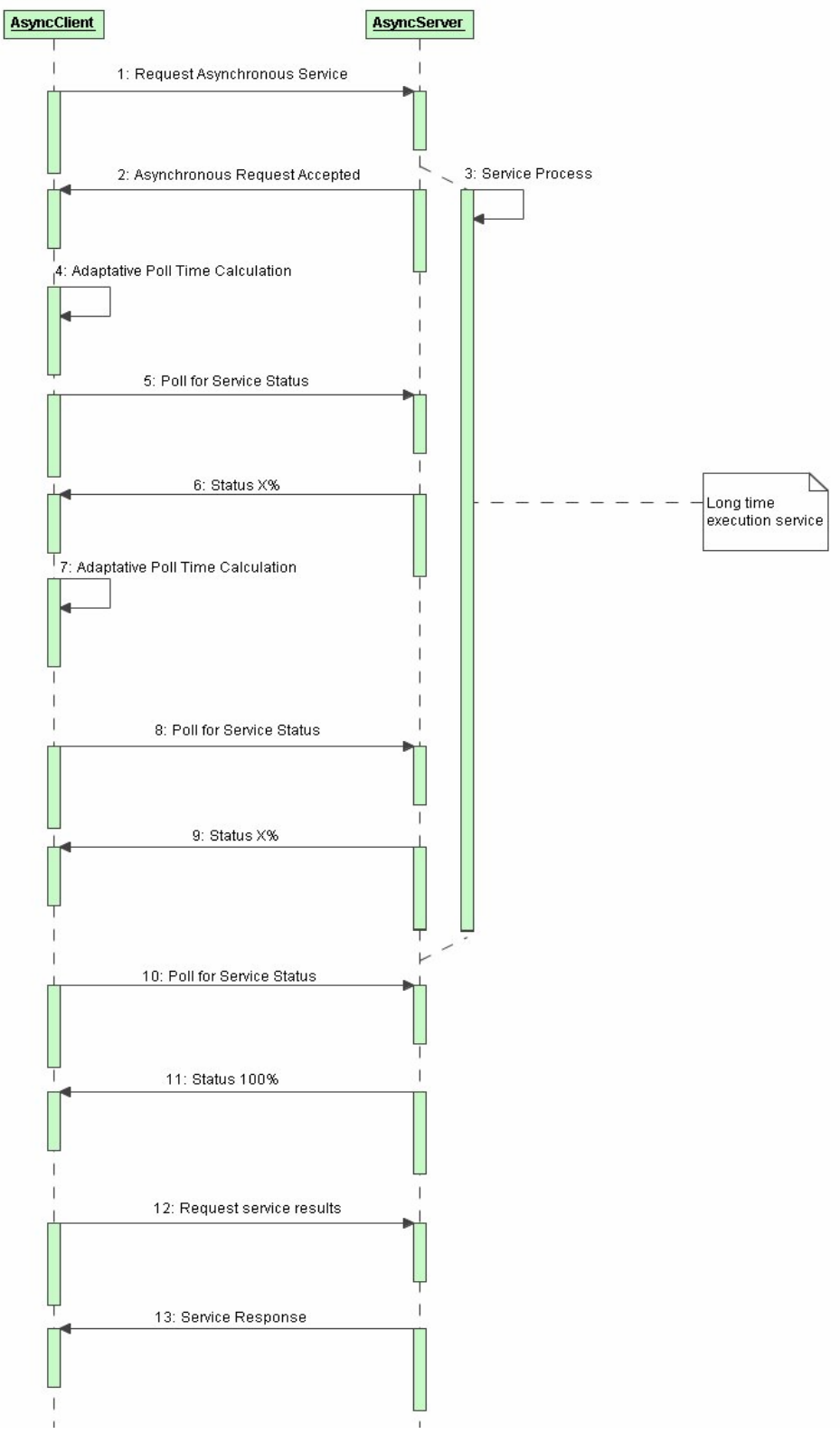
Asynchronous services will accept the task and return back a MOBY message where an identifier for the job and status information is sent back to the client, instead of the actual result. From this point both client and server will work in asynchronous modes.

C. Polling for service execution status

- The client asks the service for the status of the execution using the job identifier obtained in the previous step and the service returns a service execution status message. This step can be repeated as many times as needed until the service execution is done and the result is ready, using the ***servicename*_poll** SOAP method 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.

D. Service returns the 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 job identifier to the ***servicename*_result** SOAP method.



Sequence diagram for an async-async communication using BioMOBY

3.3. Asynchronous messages

The BioMOBY XML for service execution request and for result response will keep the original BioMOBY structure because they are data messages (containing the parameters and response sent to or received from the service).

The messages are as follows:

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

XML request for asynchronous service execution (Unmodified)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyData queryID='1'>
                        <!--- Standard BioMOBY XML for Input -->
                </moby:mobyData>
        </moby:mobyContent>
</moby:MOBY>
```

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 methods), the information that is being sent between the client and the service provider does not contain any standard BioMOBY data, but requests and information about the status of the execution. A **`mobyStatus`** block carries this kind information for each **`mobyData`** request. **`mobyStatus`** has two attributes:

   - **`queryID`**: The service provider *must* assign the same `queryID` coming from the associated `mobyData` element. This way, each `mobyStatus` is associated to one input `mobyData`
   - **`asyncID`**: The ticket representing the service provider identifier for the service execution job. The value of `asyncID` has no intrinsic meaning. The service provider should choose it be any legal XML attribute value, such that it is unique to each `mobyStatus` in the message. Clients should *not* attempt to interpret the value of `asyncID`; it is simply an identifier.

XML response for accepted asynchronous service execution (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1" asyncID="89">
                     <!—- LSAE Analysis Event Block -->
                </moby:mobyStatus>
        </moby:mobyContent>
</moby:MOBY>
```

The asynchronous choreography (poll requests and event notifications) implies the requesting of information by the client and the sending of that information from the service provider. That notification information is embedded into the `mobyStatus` tag using the OMG's LSAE standard[4] schema for Notification Events. Clients should be able to handle the five standard events defined there. If a service provider wants to define a non-standard event (allowed in the standard), then only specialized clients

[4] Life Sciences Analysis Engine (LSAE) final adopted specification - http://www.omg.org/cgi-bin/doc?dtc/2005-04-01

would be able to understand the meaning of the event. In fact, according to the LSAE standard, clients should ignore events they do not understand.

Example XML response for accepted asynchronous service execution (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1" asyncID="89">
                   <lsae:analysis_event>
                       <message>Service execution started</message>
                       <state_changed new_state="created"/>
                   </lsae:analysis_event>
                </moby:mobyStatus>
        </moby:mobyContent>
</moby:MOBY>
```

In an analogous way to the `mobyData` block, if the request of an asynchronous service fails, the XML response will be an empty `mobyStatus` block (there is the option to add a corresponding `mobyException` to describe the problem):

XML response for not accepted asynchronous service execution (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1"/>
        </moby:mobyContent>
</moby:MOBY>
```

According to the MOBY-S 0.86.3 API, exception reporting is limited to refer to mobyData. We suggest to also allow refQueryID in mobyException to be used to refer to mobyStatus elements in a similar way.

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 ***servicename*`_poll`** SOAP method in the server. Again, the MOBY XML is a polling query, and does not contain data in the normal Moby sense. The structure of the polling test message is a mobyStatus block with the asynchronous job ticket (attribute `asyncID`) and the original request query identifier (`queryID`).

XML request to poll for service execution status (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1" asyncID="89"/>
        </moby:mobyContent>
</moby:MOBY>
```

4. **Response for polling test:** The MOBY XML sent from the service is similar to that in step 2, but the `status` element includes the current process status. The `asyncID` attribute is also present, containing the ticket for the process at the service side.

XML response for polling request (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1" asyncID="89">
                   <!—- LSAE Analysis Event Block -->
                </moby:mobyStatus>
        </moby:mobyContent>
</moby:MOBY>
```

As before, The XML polling response in case of an error is an empty `mobyStatus` tag, possibly with an added `mobyException`:

XML response for polling request failure (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1"/>
        </moby:mobyContent>
</moby:MOBY>
```

5. **Requesting the result:** Once the client is notified that the service execution is completed and that the result is ready, it should call the ***servicename*_result** SOAP method, including the ticket to retrieve the final result. The message structure is the same as in the step before, but sent to a different method.

XML request for the result (New)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyStatus queryID="1" asyncID="89"/>
        </moby:mobyContent>
</moby:MOBY>
```

6. **Sending the result:** If the client requests the result from the `servicename_result` method, a standard BioMOBY response message is sent back to the client with the result of the service execution.

XML of a standard BioMOBY Service Response (Unmodified)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyData queryID='1'>
                        <!-- Standard BioMOBY XML for Output -->
                </moby:mobyData>
        </moby:mobyContent>
</moby:MOBY>
```

The XML code for the obtaining result response in case of failure will have an empty **`mobyData`** block:

XML of a failing BioMOBY Service Response (Unmodified)

```
<?xml version="1.0" encoding="UTF-8"?>
<moby:MOBY xmlns:moby='http://www.biomoby.org/moby-s'>
        <moby:mobyContent>
                <moby:mobyData queryID='1'/>
        </moby:mobyContent>
</moby:MOBY>
```