4 Using Web Services and SOAP

last updated: November 18, 2002 2:45 pm

The SOAP Conduit, part of the Modulant Contextia Interoperability Server, makes data transformations available using Web Services technology. Client programs can use SOAP messages (in XML format) as an alternative to the Interoperability API to initiate and monitor interoperability runs.

SOAP (Simple Object Access Protocol) messages, which are sent using HTTP protocol with MIME attachments, enable client programs to use TCP/IP networks, including the Internet, to communicate with Modulant Contextia Interoperability Server.

To access a Web Services/SOAP service, client programs often use an existing toolkit (from an organization such as Sun, Microsoft, Apache, etc.) to create programs that can send SOAP messages described by a WSDL (Web Services Definition Language) file published by the service. The WSDL file describes the formats of messages the service recognizes. The Modulant Contextia WSDL file is **transform.wsdl** and resides in the **conf** subdirectory of the Interoperability Server installation directory.

2.1 final

The SOAP Conduit sends a response message for each SOAP message it receives. The rest of this discussion shows examples of the types of SOAP messages that client programs can send and the format of each response message type.

This discussion includes the following sections:

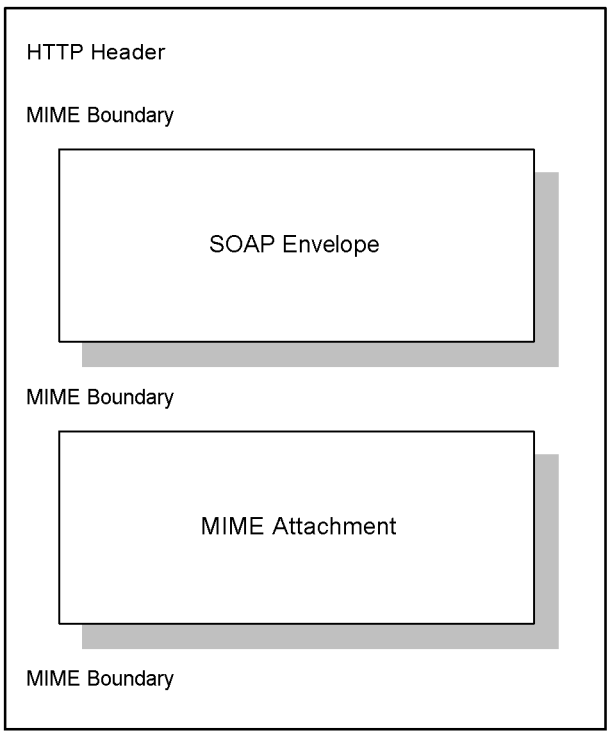
About SOAP Messages

SOAP messages use a combination of technologies:

- HTTP protocol to communicate with the Interoperability Server
- MIME encoding for sending attachments with the body of the message
- XML to define a SOAP envelope

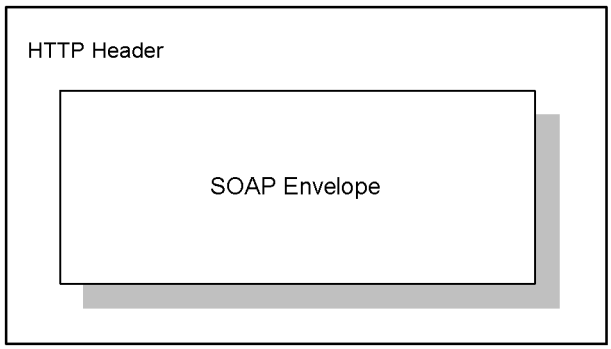
Figure 7 shows the basic format of a SOAP message with one attached file. It begins with an HTTP header, and has MIME boundaries separating the SOAP envelope and the attachment. If a message has additional attachments, they are also separated with MIME boundaries.

Figure 7: SOAP Message with Attached File



If a SOAP message has no attachments, it contains only the HTTP header and the SOAP envelope. Figure 8 shows how the basic structure is simplified when there are no attached files.

Figure 8: SOAP Message with No Attachments



The remaining sections of this chapter show the SOAP envelope section of each message type the Interoperability Server recognizes. Messages that can have attachments show the first few lines of the MIME boundary to indicate the presence of at least one attached file.

For more information and examples of each message type:

- Uploading Files to the Interoperability Server
- Initiating Synchronous Interoperability Runs
- Submitting Asynchronous Interoperability Jobs
- Getting the Status and Results of a Queued Job
- Managing Queued Interoperability Jobs
- TransformException and SOAP Errors

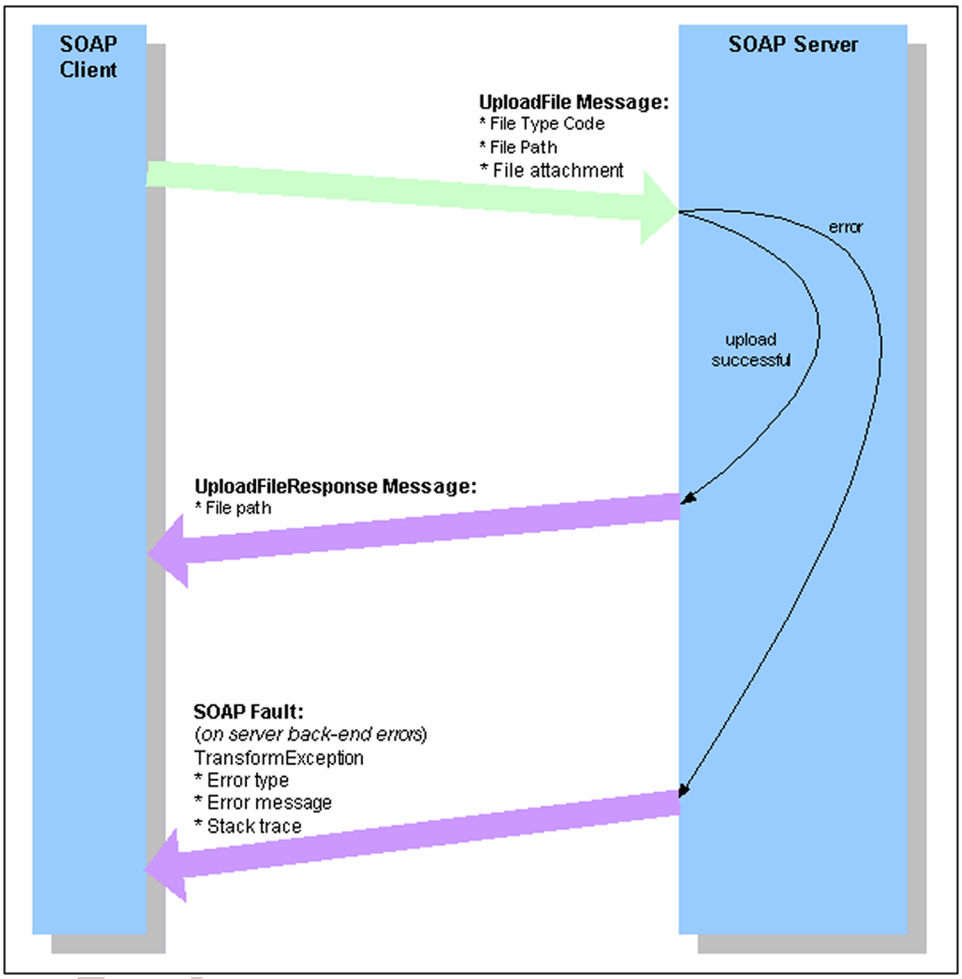
2.1 final

Uploading Files to the Interoperability Server

You can use SOAP messages to send files to be used in interoperability runs to a remote Interoperability Server at any time. To do this, a client program creates an **uploadFile** message and attaches the file to upload. This message accepts one attachment at a time.

Figure 9 shows how a SOAP client uploads a file to a SOAP server.

Figure 9: Uploading Files to Interoperability Server Cache



Example 19 shows the format of the **uploadFile** message. This message includes all of the run configuration parameters for this interoperability run.

The **uploadFile** message expects one file to be attached. For each attachment, you must put the filename from the **filePath** entry into the **content-location** tag of the MIME header for that attachment. The path information points to the client's file system.

Example 19: uploadFile Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
       <ns1:uploadFile xmlns:ns1="http://modulant.com/transform">
          <fileType>4</fileType>
          <filePath>..\ats\acmeauto.xml</filePath>
       </ns1:uploadFile>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_7756310.1020465834308
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <1006561434.1020465834138.AXIS@BTAN-DT>
content-location: ..\ats\acmeauto.xml
. . .
```

To specify the value of the **fileType** element, use one of the numeric values in Table 5 on page 124.

If the specified file is not attached, you get a failure message instead of the standard **uploadFileResponse** message.

The **uploadFile** message is equivalent to using the **uploadFile()** method of the **Transformation** class in the Interoperability API; for more information, see "Upload.uploadFile()" on page 124.

Example 20 shows the format of the **uploadFileResponse** message. This message contains a **filePath** entry indicating where the Interoperability Server put the uploaded file.

Example 20: uploadFileResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
       <uploadFileResponse>
          <filePath>..\ats\acmeauto.xml</filePath>
```

2.1 final

```
        </uploadFileResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Initiating Synchronous Interoperability Runs

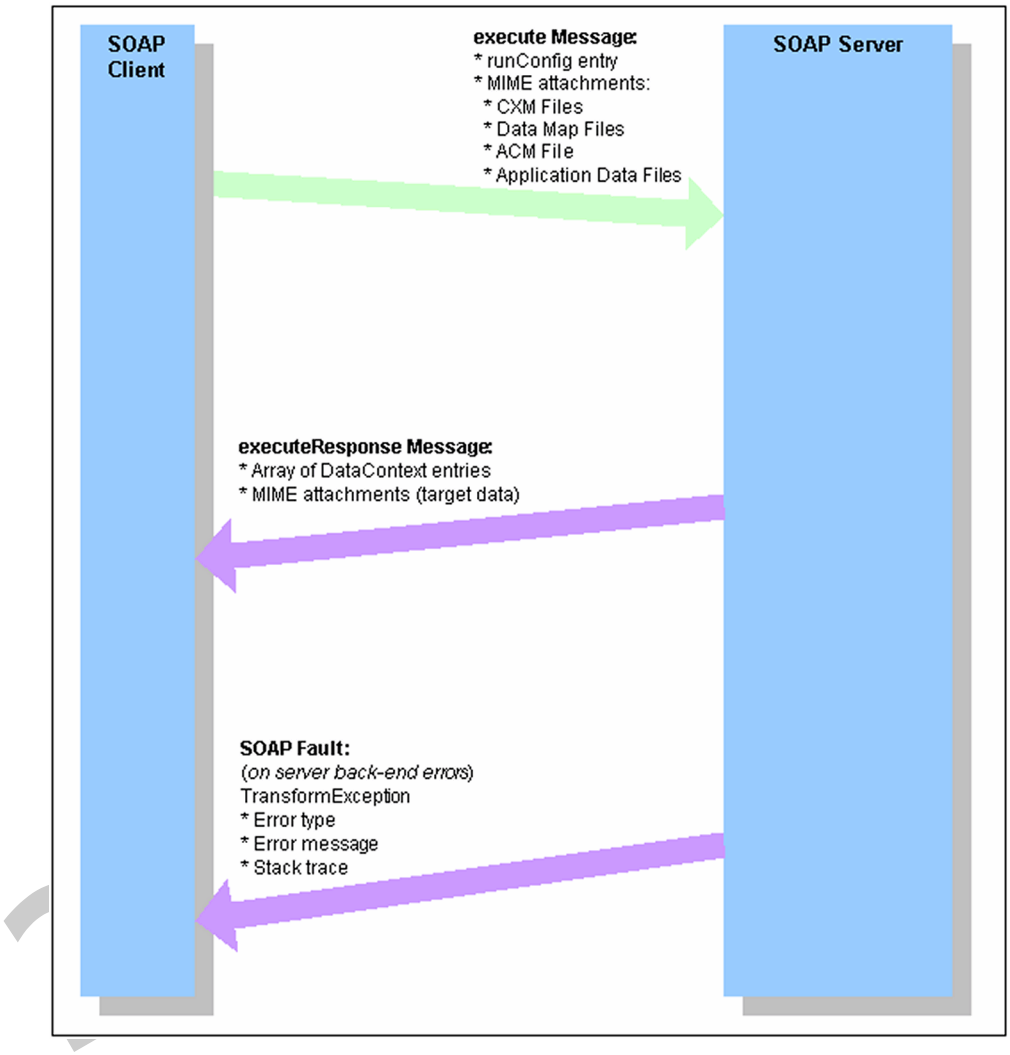
To request a synchronous interoperability run, a client program creates an **execute** message that contains the run configuration information, and attaches the required files.

For details about the format of the run configuration parameters, see Appendix A, "run_config.xml Reference." For more information about the files required for an interoperability run, see "Specifying Run Configuration Parameters" on page 19.

Figure 10 shows how a SOAP client communicates with a SOAP server.

2.1 final

Figure 10: Requesting Synchronous Interoperability Runs



Example 21 shows the format of the **execute** message. This message includes all of the run configuration parameters for this interoperability run. The body of the **execute** message must contain a **RunConfig** complex type as defined in the **transform.wsdl** file.

The **execute** message also requires the necessary files to be attached. Whenever a client program uploads files with the **execute** request, the Interoperability Server ignores files that are already available on the server. For each attachment, you must put the filename from the **runConfig** section into the **content-location** tag of the MIME header for that attachment.

Example 21: execute Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:execute xmlns:ns1="http://modulant.com/transform">
         <runConfig><description>AcmeAuto test</description>
            <acm>acm\abstract_conceptual_model_B.EXP</acm>
            <enforceUnique>true</enforceUnique>
            . . .
         </runConfig>
      </ns1:execute>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_2736286.1020111692751
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <713854469.1020111692620.AXIS@BTAN-DT>
content-location: acm\abstract_conceptual_model_B.EXP
. . .
```

The **execute** message is equivalent to using the **execute()** method of the **Transformation** class in the Interoperability API; for more information, see "Transformation.execute()" on page 101.

If any of the required files are not attached or available on the server machine, you get a failure message instead of the standard **executeResponse** message.

Example 22 shows the format of the **executeResponse** message. This message contains one **DataContext** complex type for each **target** entry in the run configuration parameters in the original **execute** message.

The format of the returned **DataContext** entry is described by the **DataContext** complex type in the **transform.wsdl** file. The response message contains one attachment for each **target** entry that expects application data in a file. For each attachment, the filename in the **content-location** tag of the MIME header matches matches the filename in the **DataContext** entry.

For more information, see "DataContext Interface" on page 67.

Example 22: executeResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <executeResponse>
         <DataContext>
            <cxm>C:\Modulant\cxm\acmeauto.cxm</cxm>
            <contextMapName>Test</contextMapName>
            <dataMap>C:\Modulant\ats\acme_datamap.xml</dataMap>
            <dataLocation>file</dataLocation>
```

2.1 final

```
                <applicationData>
                    C:\Modulant\samples\AcmeAuto\acmeauto.xml
                </applicationData>
            </DataContext>
        </executeResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_2473302.1020111710486
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <933472107.1020111710466.AXIS@BTAN-DT>
content-location: C:\Modulant\samples\AcmeAuto\acmeauto-out.xml
. . .
```

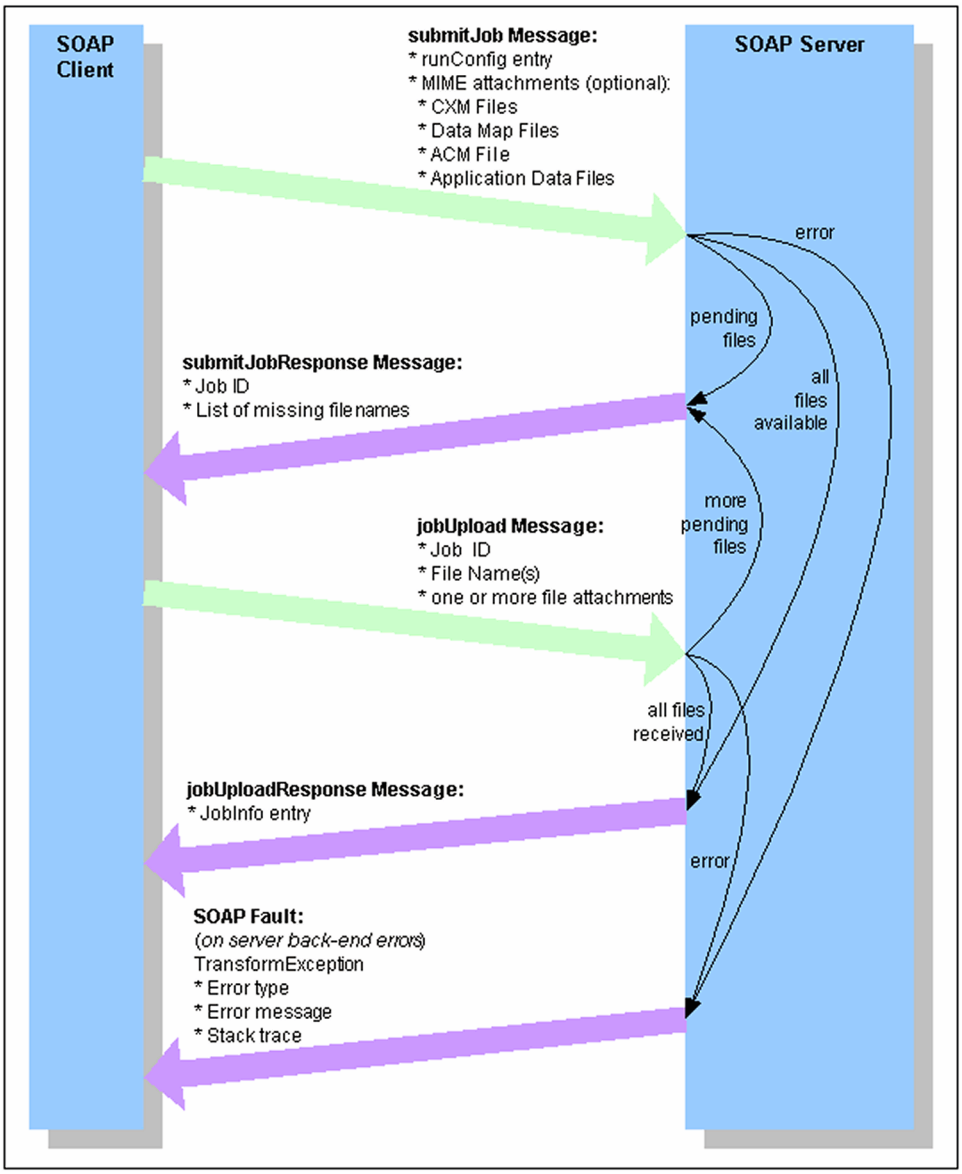
Submitting Asynchronous Interoperability Jobs

To request an asynchronous interoperability run, a client program creates a **submitJob** message that contains the run configuration information, and attaches the required files.

Figure 11 shows how a SOAP client communicates with a SOAP server to send asynchronous interoperability run requests and the associated files to the job queue.

2.1 final

Figure 11: Submitting Asynchronous Interoperability Jobs



This section describes the following message types:

- Submitting Job Requests
- Uploading Job Files

Submitting Job Requests

Example 23 shows the format of the **submitJob** message. This message includes all of the run configuration parameters for this interoperability run. The body of the **submitJob** message must contain a **RunConfig** complex type as defined in the **transform.wsdl** file.

The **submitJob** message can also have the necessary files attached. Whenever a client program uploads files with the **submitJob** request, the Interoperability Server ignores files that are already available on the server. For each attachment, you must put the filename from the **runConfig** section into the **content-location** tag of the MIME header for that attachment.

For details about the format of the run configuration parameters, see Appendix A, "run_config.xml Reference."

For more information about the files required for an interoperability run, see "Specifying Run Configuration Parameters" on page 19.

Example 23: submitJob Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
       <ns1:submitJob xmlns:ns1="http://modulant.com/transform">
          <runConfig><description>AcmeAuto test</description>
             <acm>data\abstract_conceptual_model_B.EXP</acm>
             <enforceUnique>true</enforceUnique>
             . . .
          </runConfig>
       </ns1:submitJob>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_2736286.1020111692751
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <713854469.1020111692620.AXIS@BTAN-DT>
content-location: acm\abstract_conceptual_model_B.EXP
. . .
```

The **submitJob** message is equivalent to using the **submitJob()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.submitJob()" on page 111.

Example 24 shows the format of the **submitJobResponse** message. This message contains a **JobInfo** complex type with information about the queued interoperability job. The format of the returned **JobInfo** entry is described by the **JobInfo** complex type in the **transform.wsdl** file.

For more information, see "JobInfo Interface" on page 77.

2.1 final

Example 24: submitJobResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
       <submitJobResponse>
           <JobInfo>
               <jobId>2189</jobId>
               <description>AcmeAuto test</description>
               <status>ASSEMBLING</status>
               <statusMessage/>
               <pendingFiles>
                   <file>samples\AcmeAuto\acmeauto.xml</file>
               </pendingFiles>
               <submitTime>2002-04-29T20:31:43.621Z</submitTime>
               <startTime/>
               <endTime/>
           </JobInfo>
       </submitJobResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Notice the **pendingFiles** entry in the **JobInfo** object. This contains a list of files that the Interoperability Server could not find. You can send a separate **jobUpload** message with the missing files attached. This message is described in "Uploading Job Files."

Uploading Job Files

When you send a **submitJob** message, the SOAP server might not receive all of the files it needs to queue the interoperability job. When the Interoperability Server indicates that some of the necessary files are still missing, a SOAP client can provide those missing files using a **jobUpload** message.

Example 25 shows the format of the **jobUpload** message. This message contains the job number that the Interoperability Server returned in the **submitJobResponse** message, and has the missing files attached in MIME format.

Whenever a client program uploads files with the **jobUpload** request, the Interoperability Server ignores files that are already available on the server. For each attachment, you must put the filename from the **runConfig** section for the job you reference into the **content-location** tag of the MIME header for that attachment. The path information points to the client's file system.

For details about the format of the run configuration parameters, see Appendix A, "run_config.xml Reference."

For more information about the files required for an interoperability run, see "Specifying Run Configuration Parameters" on page 19.

2.1 final

Example 25: jobUpload Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:jobUpload xmlns:ns1="http://modulant.com/transform">
         <jobId>1413</jobId>
      </ns1:jobUpload>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_2736286.1020111692751
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <713854469.1020111692620.AXIS@BTAN-DT>
content-location: acm\abstract_conceptual_model_B.EXP
. . .
```

The **jobUpload** message is equivalent to using the **uploadFile()** method of the **TransformationJob** class in the Interoperability API; for more information, see "Upload.uploadFile()" on page 124.

Example 26 shows the format of the **jobUploadResponse** message. This message includes a **JobInfo** complex type with a list of pending files. The format of the returned **JobInfo** entry is described by the **JobInfo** complex type in the **transform.wsdl** file.

Example 26: jobUploadResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <jobUploadResponse>
         <JobInfo>
            <jobId>2189</jobId>
            <description>AcmeAuto test</description>
            <status>QUEUED</status>
            <statusMessage/>
            <pendingFiles/>
            <submitTime>2002-04-29T20:31:43.000Z</submitTime>
            <startTime/>
            <endTime/>
         </JobInfo>
      </jobUploadResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

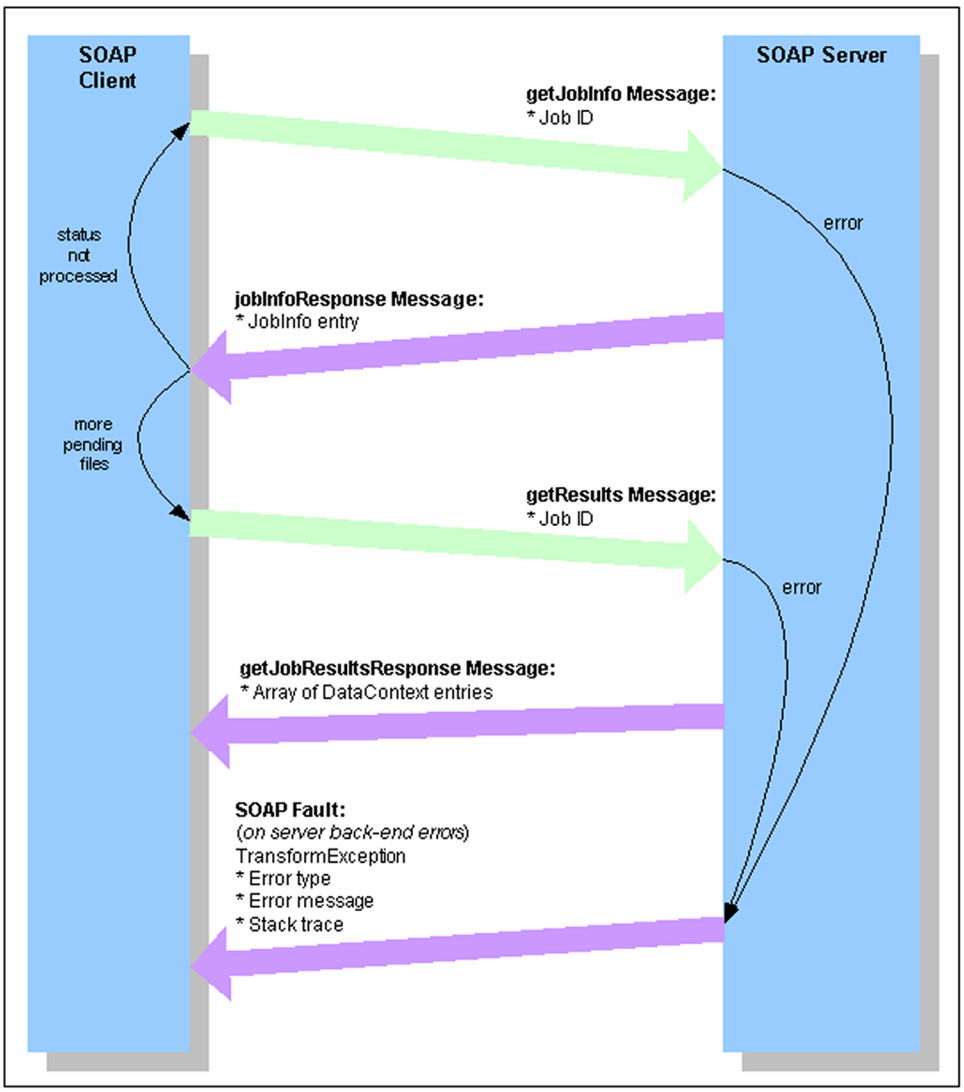
2.1 final

Getting the Status and Results of a Queued Job

Once you have added an interoperability job to the queue, you can request information about its status. After the job has successfully completed execution, you can request to have the target data returned from the Interoperability Server to the locations specified in the run configuration parameters.

Figure 12 shows how a SOAP client communicates with a SOAP server to request the status of a queued job and to request target data after a successful interoperability run.

Figure 12: Getting the Status and Results of a Queued Job



This section describes the following message types:

- Getting Information About a Job
- Retrieving the Target Data from an Interoperability Run

Getting Information About a Job

To get information about a job, a client program creates a **getJobInfo** message. Example 27 shows the format of the **getJobInfo** message. This message contains the job number that was returned in the **submitJobResponse** message.

Example 27: getjobInfo Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:getJobInfo xmlns:ns1="http://modulant.com/transform">
         <jobId>1413</jobId>
      </ns1:getJobInfo>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The **getJobInfo** message is equivalent to using the **getJobInfo()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.getJobInfo()" on page 106.

Example 28 shows the format of the **getJobInfoResponse** message. This message contains a **JobInfo** complex type with information about the queued interoperability job. The format of the returned **JobInfo** entry is described by the **JobInfo** complex type in the **transform.wsdl** file.

For more information, see "JobInfo Interface" on page 77.

Example 28: getJobInfoResponse Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <getJobInfoResponse>
         <JobInfo>
            <jobId>2190</jobId>
            <description>AcmeAuto test</description>
            <status>PROCESSING</status>
            <statusMessage>Extraction</statusMessage>
            <pendingFiles/>
            <submitTime>2002-04-29T21:29:26.000Z</submitTime>
            <startTime>2002-04-29T21:29:29.000Z</startTime>
            <endTime/>
         </JobInfo>
      </getJobInfoResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.1 final

Note: The times in this response message (in the **submitTime**, **startTime**, and **endTime** elements) are in GMT (Greenwich Mean Time). You must convert them to your own time zone yourself.

Retrieving the Target Data from an Interoperability Run

To retrieve the target application data produced as the result of an interoperability run, a client program creates a **getJobResults** message. Example 29 shows the format of the **getJobResults** message. This message contains the job number that was returned in the **submitJobResponse** message.

Example 29: getjobResults Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:getJobResults
             xmlns:ns1="http://modulant.com/transform">
         <jobId>1413</jobId>
      </ns1:getJobResults>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The **getJobResults** message is equivalent to using the **getJobResults()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.getJobResult()" on page 106.

If you send a **getJobResults** message and the job has not finished running, you get a SOAP **Fault** message with a **TransformException** in response. For more information, see "TransformException and SOAP Errors" on page 63.

Example 30 shows the format of the **getJobResultsResponse** message. This message contains one **DataContext** complex type for each **target** entry in the run configuration parameters in the original **execute** message.

The format of the returned **DataContext** entry is described by the **DataContext** complex type in the **transform.wsdl** file. The response message contains one attachment for each **target** entry that expects application data in a file. For each attachment, the filename in the **content-location** tag of the MIME header matches matches the filename in the **DataContext** entry.

For more information, see "DataContext Interface" on page 67.

2.1 final

Example 30: getJobResultsResponse Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <getJobResultsResponse>
         <DataContext>
            <cxm>C:\Modulant\cxm\acmeauto.cxm</cxm>
            <contextMapName>Test</contextMapName>
            <dataMap>C:\Modulant\ats\acme_datamap.xml</dataMap>
            <dataLocation>file</dataLocation>
            <applicationData>
               C:\Modulant\samples\acmeauto\acme.xml
            </applicationData>
         </DataContext>
      </getJobResultsResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Managing Queued Interoperability Jobs

Using SOAP messages, you can request information about queued jobs with a particular status. You can also cancel running jobs and you can remove jobs from the queue.

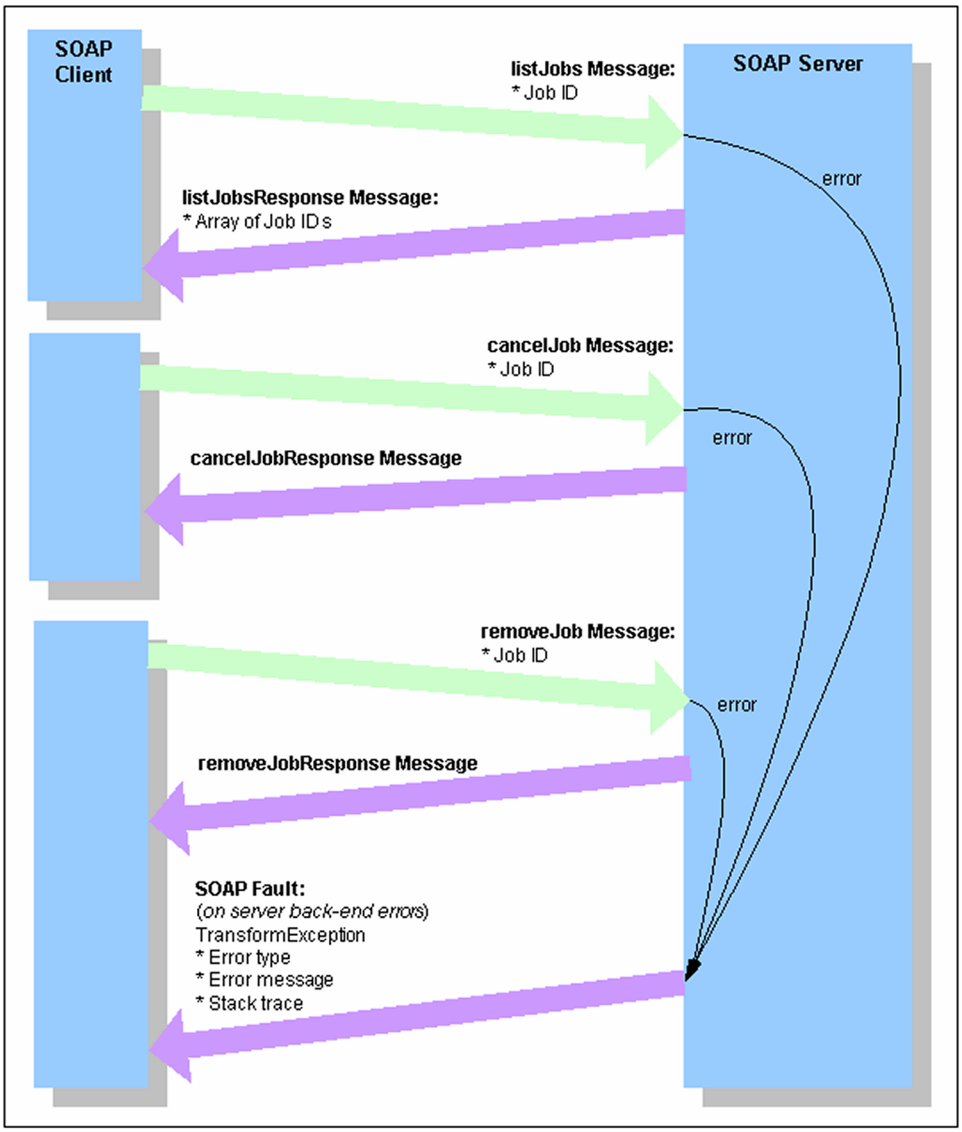
Figure 13 shows how a SOAP client communicates with a SOAP server to list, cancel, or remove queued jobs.

This section describes the following message types:

- Listing Queued Jobs with a Specific Status
- Cancelling Queued Jobs
- Removing Jobs from the Queue

2.1 final

Figure 13: Managing Queued Interoperability Jobs



Listing Queued Jobs with a Specific Status

To get a list of queued interoperability jobs with a particular status, a client program creates a **listJobs** message.

Example 31 shows the format of the **listJobs** message. This message contains the numeric value of the status you want to query. Table 2 on page 40 shows a list of possible status values.

Example 31: listJobs Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:listJobs xmlns:ns1="http://modulant.com/transform">
         <status>3</status>
      </ns1:listJobs>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The **listJobs** message is equivalent to using the **listJobs()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.listJobs()" on page 109.

Example 32 shows the format of the **listJobsResponse** message. This message contains a list of job numbers corresponding to jobs with the status value in the **listJobs** message.

Example 32: listJobsResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <listJobsResponse>
         <jobId>2184</jobId>
         <jobId>2185</jobId>
         <jobId>2187</jobId>
         <jobId>2189</jobId>
         <jobId>2168</jobId>
         <jobId>2169</jobId>
         <jobId>2171</jobId>
      </listJobsResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2-1 final

Cancelling Queued Jobs

To cancel a running interoperability job, a client program creates a **cancelJob** message. Example 33 shows the format of the **cancelJob** message. This message contains the job number that was returned in the **submitJobResponse** message.

Example 33: cancelJob Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
   <SOAP-ENV:Body>
      <ns1:cancelJob xmlns:ns1="http://modulant.com/transform">
         <jobId>1413</jobId>
      </ns1:cancelJob>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The **cancelJob** message is equivalent to using the **cancelJob()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.cancelJob()" on page 105.

Example 34 shows the format of the **cancelJobResponse** message. This message contains the number of the job that was cancelled.

Example 34: cancelJobResponse Message

```
<SOAP-ENV:Envelope
       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Body>
        <cancelJobResponse>
            <jobId>1413</jobId>
        </cancelJobResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Removing Jobs from the Queue

To remove a job from the queue, a client program creates a **removeJobs** message. Example 35 shows the format of the **removeJob** message. This message contains the job number that was returned in the **submitJobResponse** message.

Example 35: removeJob Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
       "http://schemas.xmlsoap.org/soap/envelope/". . .>
    <SOAP-ENV:Body>
        <ns1:removeJob xmlns:ns1="http://modulant.com/transform">
            <jobId>2189</jobId>
        </ns1:removeJob>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The **removeJob** message is equivalent to using the **removeJob()** method of the **TransformationJob** class in the Interoperability API; for more information, see "TransformationJob.removeJob()" on page 110.

Example 36 shows the format of the **removeJobResponse** message. This message contains the number of the job that was removed from the queue.

2.1 final

Example 36: removeJobResponse Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <removeJobResponse>
         <jobId>2189</jobId>
      </removeJobResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

TransformException and SOAP Errors

If a SOAP request fails, the SOAP server returns a **Fault** message. This message type can replace any expected response message when an error occurs. Every method in the Interoperability API that can generate a **TransformException** can reply with a SOAP **Fault** message.

Example 37 shows a **getJobInfo** request failing with a SOAP **Fault** message reply. This message includes a copy of the stack trace generated by the Interoperability Server.

2.1 final

Example 37: Fault Message

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode xmlns:ns1="http://xml.apache.org/axis/">
            ns1:Server.userException
         </faultcode>
         <faultstring>
            modulant.transform.api.exception.TransformException:
               no data found for id: 5555
         </faultstring>
         <detail>
            <ns2:stackTrace
                  xmlns:ns2="http://xml.apache.org/axis/">
               . . .
               (TRACE MESSAGES)
               . . .
            </ns2:stackTrace>
         </detail>
   </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.1 final