JobScheduler Web Services

Executing JobScheduler commands

Technical Reference

March 2015

Contact Information

Software- und Organisations-Service GmbH

Giesebrechtstr. 15
D-10629 Berlin
Germany

Telephone +49 (0)30 86 47 90-0
Telefax +49 (0)30 8 61 33 35
Mail info@sos-berlin.com
Web http://www.sos-berlin.com

Last Updated: 03/13/2015 12:01 PM

This documentation is based on JobScheduler Version 1.7.4169.

This product includes software developed by the Apache Software Foundation (http://apache.org/)

We would appreciate any feedback you have, or suggestions for changes and improvements; please forward your comments to info@sos-berlin.com.

Table of Contents

1 Abstract

... to be defined ...

2 Getting started

... to be defined ...

3 Introduction

The JobScheduler offers an interface with which the JobScheduler "external API" are provided as Web Services. With the external API objects, for example jobs, job chains and orders, can be scheduled. The following two types of Web Service can be provided:

Asynchronous Web Services

The JobScheduler can be configured to accept and reply to Web Service requests using XSLT-Stylesheets . It is not restricted to using a specialized protocol above HTTP (such as SOAP). The implementation of this protocol is not contained in the JobScheduler itself but in the style sheets.

Existing job chains can be adapted with this method for use as Web Services. The use of this technique means that the results from order processing are relayed asynchronously. A synchronous response is however also possible with this method - as confirmation that an order has been accepted.

Synchronous Web Services

Should synchronous responses containing the results of job processing be required, then the necessary interfaces must be included in the jobs themselves. Synchronous job responses are then sent by way of the JobScheduler API.

The Web Service queries are then interpreted in jobs. Style sheets are not used - instead, a job chain is configured to reply to the Web Service.

An understanding of the following subjects is seen as a requirement for this tutorial:

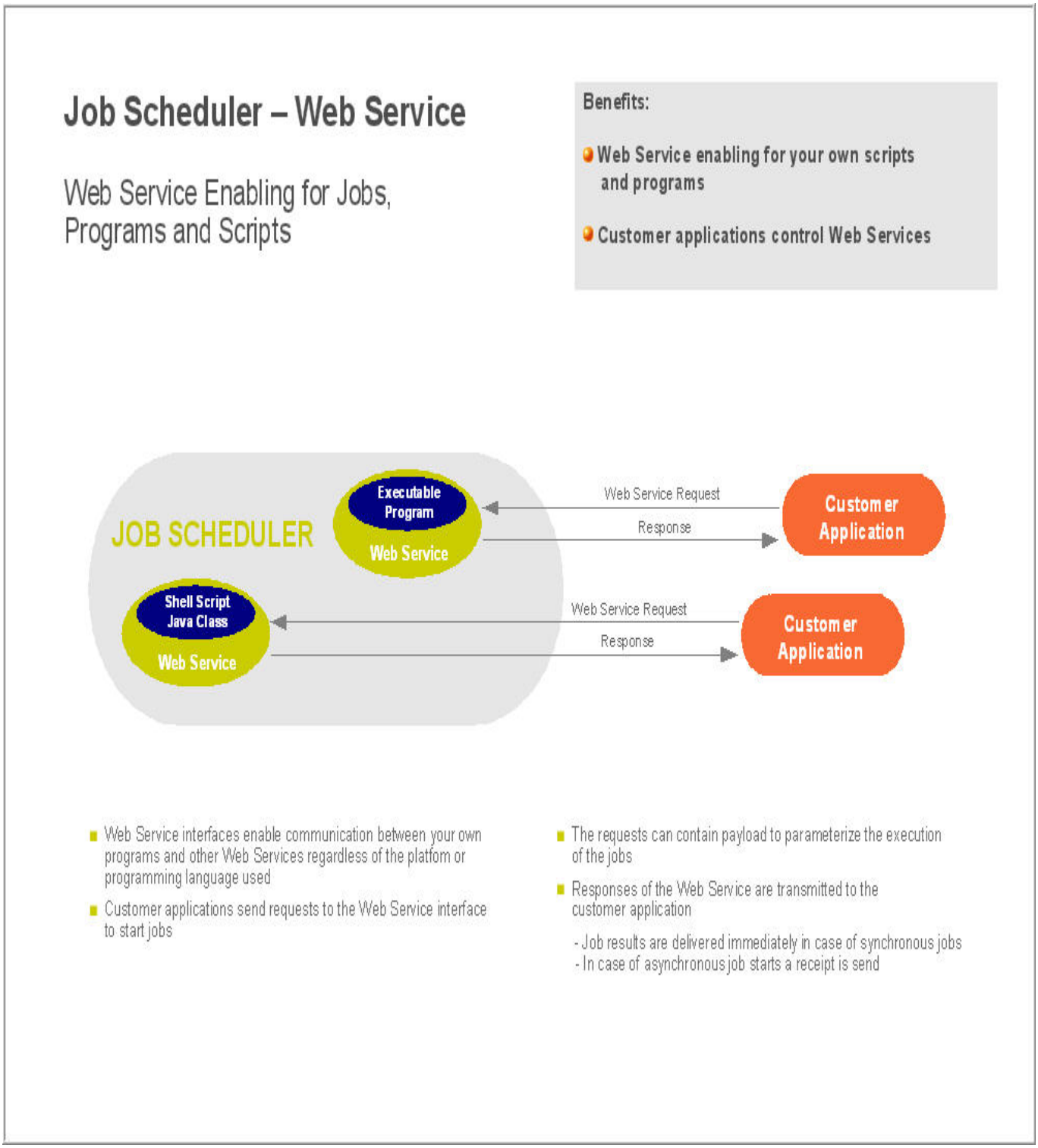
- Knowledge of the syntax of SOAP messages
- An understanding of the use of Jobs and Job Chains in the JobScheduler
- Basic knowledge of the JobScheduler XML Commands
- Basic knowledge of XSLT (for asynchronous Web Services)
- Knowledge of the JobScheduler internal API (for example Java) for the job implementation (for synchronous Web Services)

4 Use of the JobScheduler for Web Services

The JobScheduler can make objects available to start initiated by a Web Service. The following examples show the usage of this interface in different fields of application

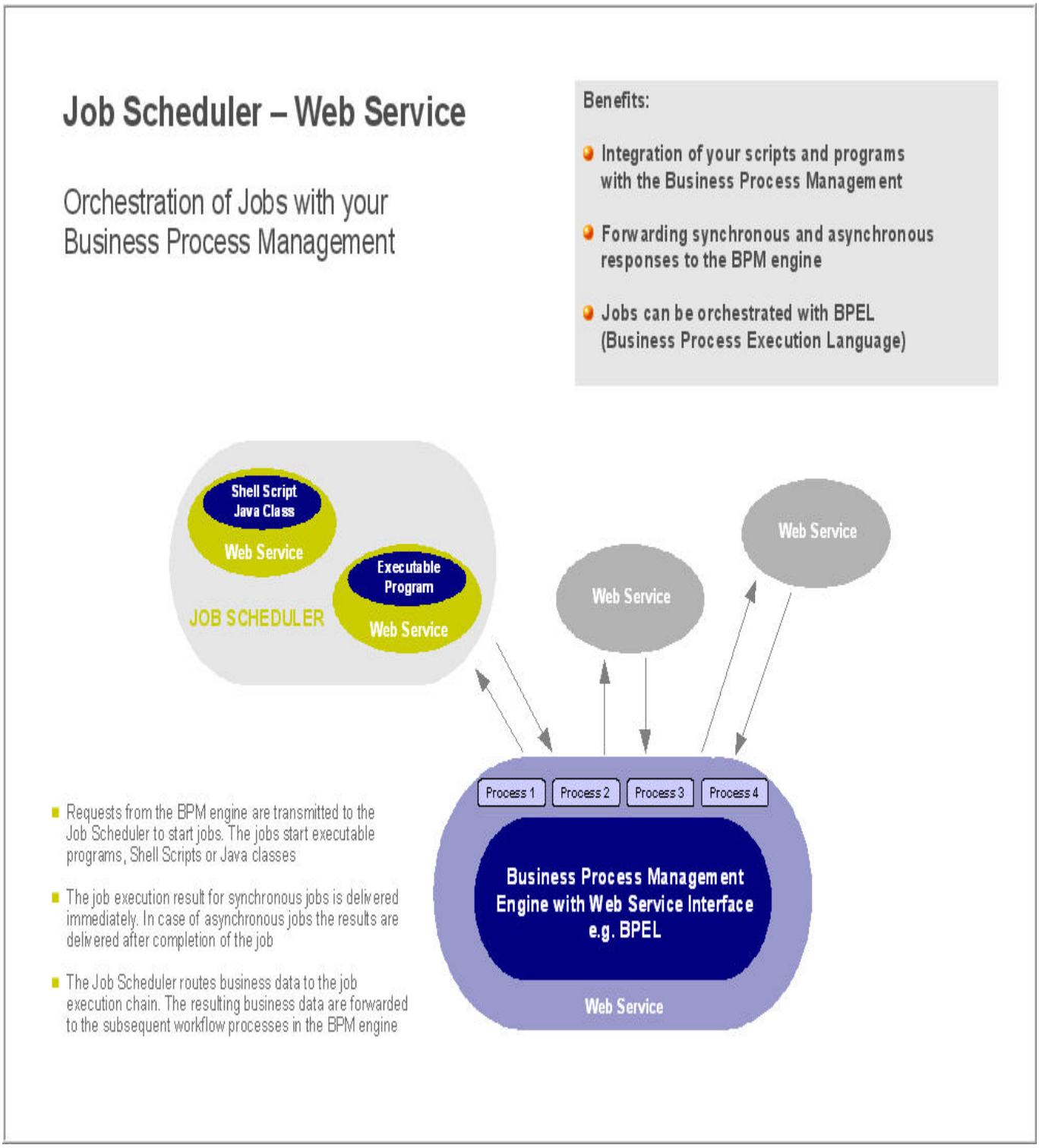
4.1 The Web Service Capability of Programs and Scripts

Executable programs und shell scripts can be made available as Web Services. Applications can then control JobScheduler objects by way of Web Service calls. In turn, these objects can be used to start programs, (shell-) scripts or Java classes.



4.2 Jobs Orchestration using Web Services

Programs and scripts can be integrated in Business Process Management (BPM) using the JobScheduler Web Service. The JobScheduler then delivers synchronous and asynchronous responses to the BPM engine.



4.3 Routing Web Service Requests

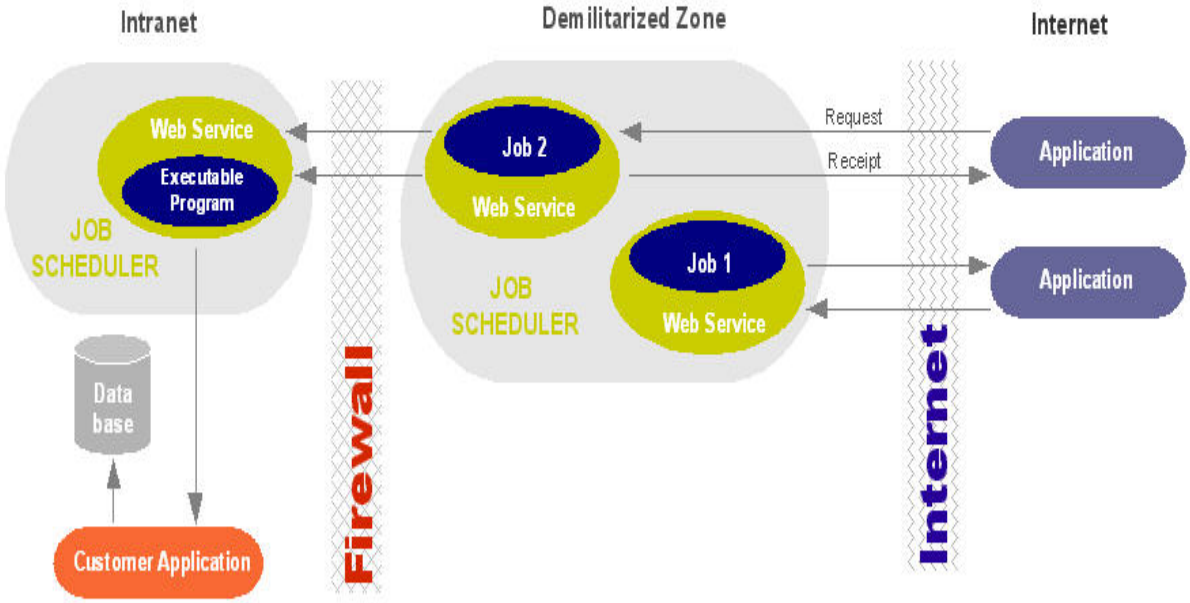
The JobScheduler can route Web Service requests from the internet to an intranet. The requests are then received, authenticated and validated, and valid requests forwarded to a further JobScheduler in the intranet.

Job Scheduler – Web Service

Routing of Web Service Requests

Benefits:

- Route requests from the internet to the intranet
- Receive, authenticate and validate Web Service requests from the internet
- Forward valid requests to the Job Scheduler in the intranet for further processing



- The Job Scheduler can be deployed within the demilitarized zone without additional components such as servlet container or database to minimize security risks.
- The Job Scheduler authenticates the IP address of the incoming Web Service request
- The payload of the request is validated by means of XML Schema
- The request is forwarded to a second Job Scheduler in the intranet that starts programs and scripts of the customer application
- Immedeately the Job Scheduler sends receipts for valid requests and error messages for invalid requests
- Job results from the job processing are returned asynchronously to the application that originated the request

5 The Asynchronous Web Service Query Process

The JobScheduler must be configured to provide a Web Service by way of an URL. This URL is then when a client sends a Web Services query to the JobScheduler (host and TCP port) per HTTP POST. In the example presented in this documentation the `http://localhost:4444/execute_file` URL is used.

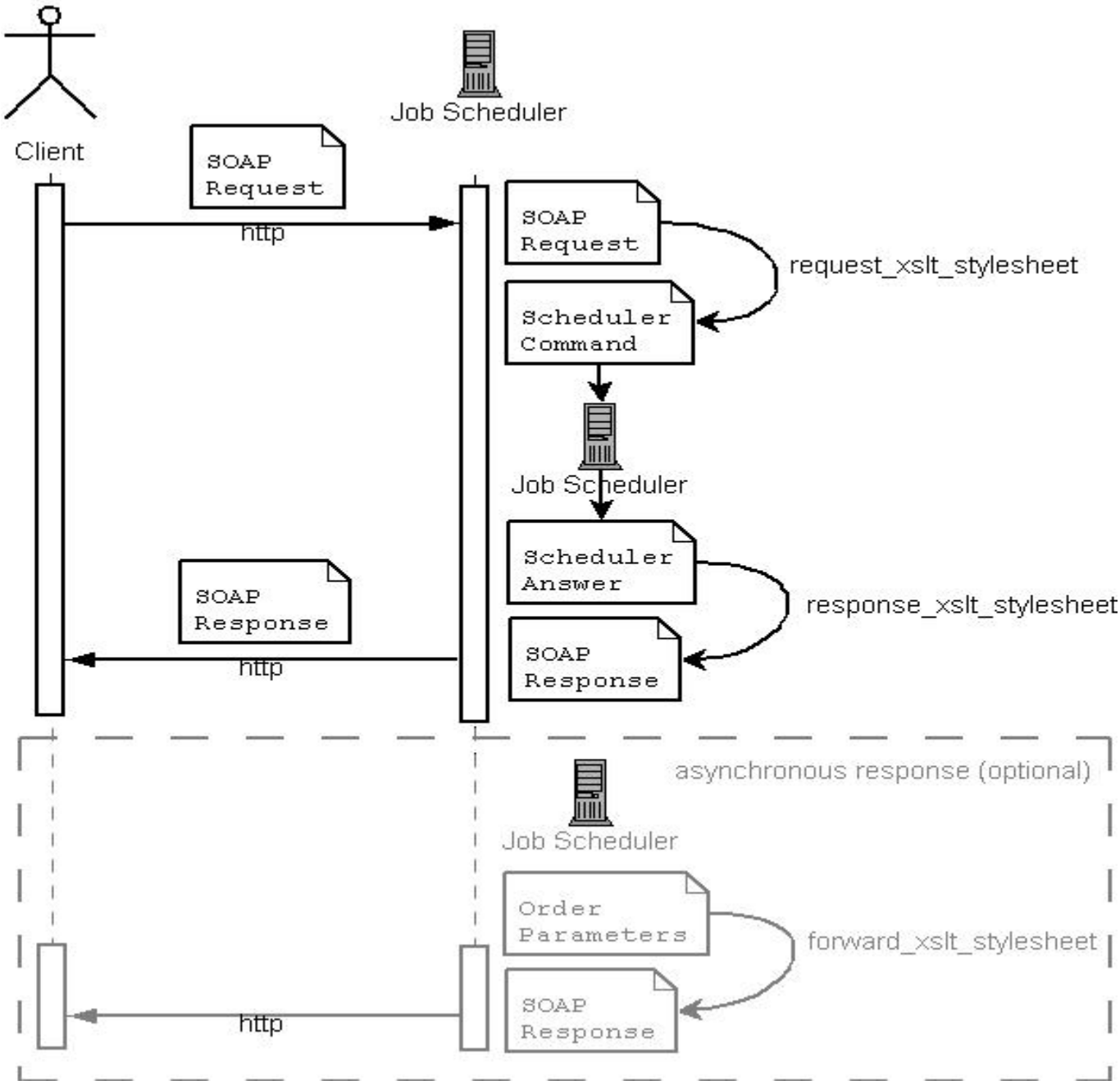
The JobScheduler is not able to understand such incoming queries directly and therefore uses a request_xslt_stylesheet which has been preconfigured for the Web Service. This style sheet is used to transform the query into an XML command which can then be interpreted by the JobScheduler .

The JobScheduler executes the XML command and holds the HTTP connection open. Further, it returns answers in XML format after executing XML commands. For example, the JobScheduler returns the following XML answer on successful completion of the `<kill_task>` command:

```
<spooler>
<answer>
<ok/>
</answer>
</spooler>
```

The (SOAP) client at the other end of the open HTTP connection expects a synchronous reply in the form of a SOAP message. The JobScheduler therefore uses the response_xslt_stylesheet to transform its XML responses into a format which the client can understand. After sending this reply to the client, the JobScheduler closes the HTTP connection.

Such synchronous responses generally contain relatively simple information such as whether, for example, an order has been successfully received and which ID it has been allocated. In order to provide a Web Service in which the response contains the results of a completed order, it must also be possible to make responses asynchronously. This is necessary as order queuing and execution can take longer than the time for which the HTTP connection is kept alive. In this case, the order parameters (which change as a job chain is processed) are transformed using the style sheet "forward_xslt_stylesheet" . The result of the order is sent as an asynchronous response to the (SOAP) client (or a completely different recipient) using a new HTTP connection.

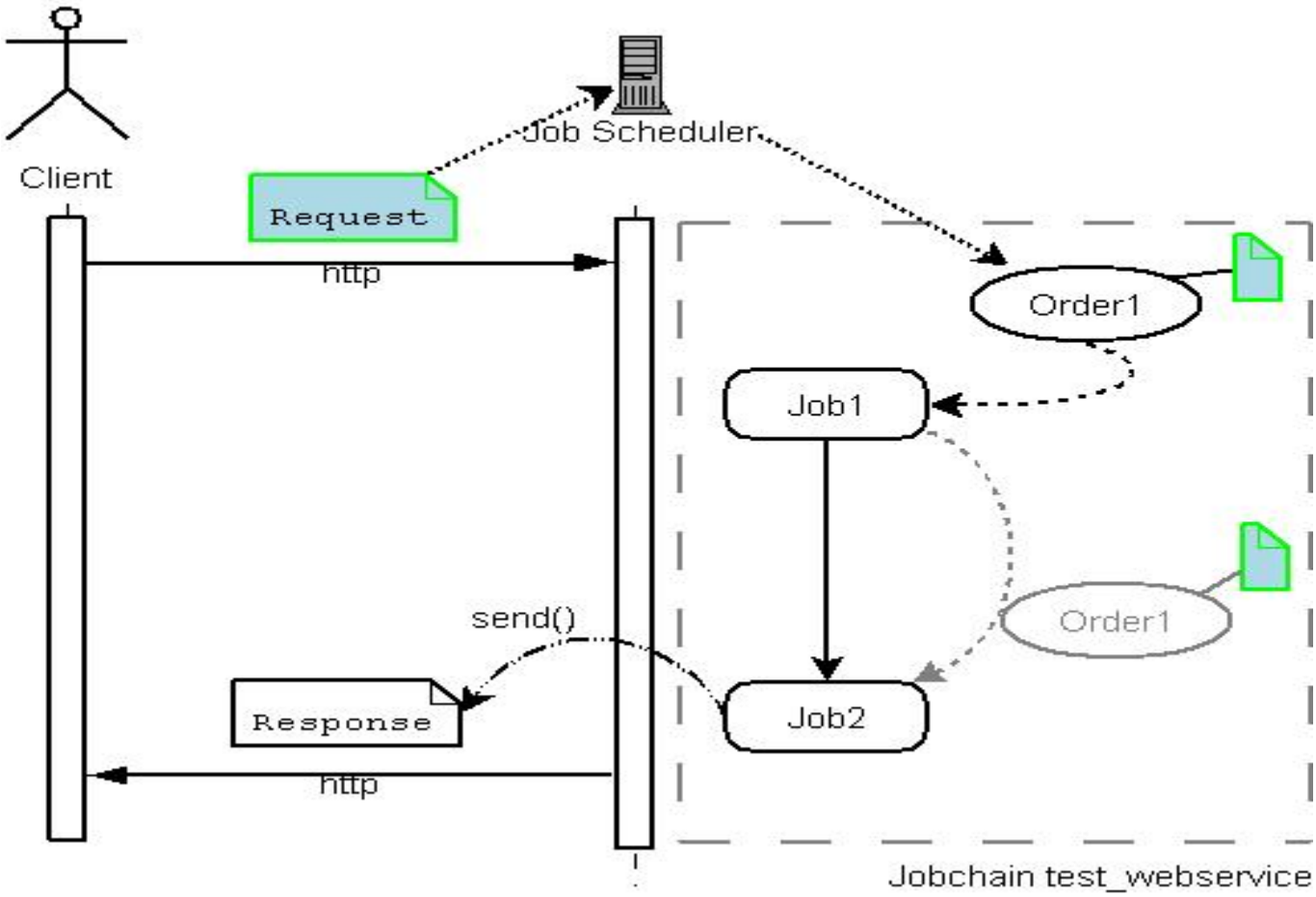

Client
Job Scheduler
SOAP Request
http
SOAP Request
request_xslt_stylesheet
Scheduler Command
Job Scheduler
Scheduler Answer
response_xslt_stylesheet
SOAP Response
SOAP Response
http
asynchronous response (optional)
Job Scheduler
Order Parameters
forward_xslt_stylesheet
SOAP Response
http

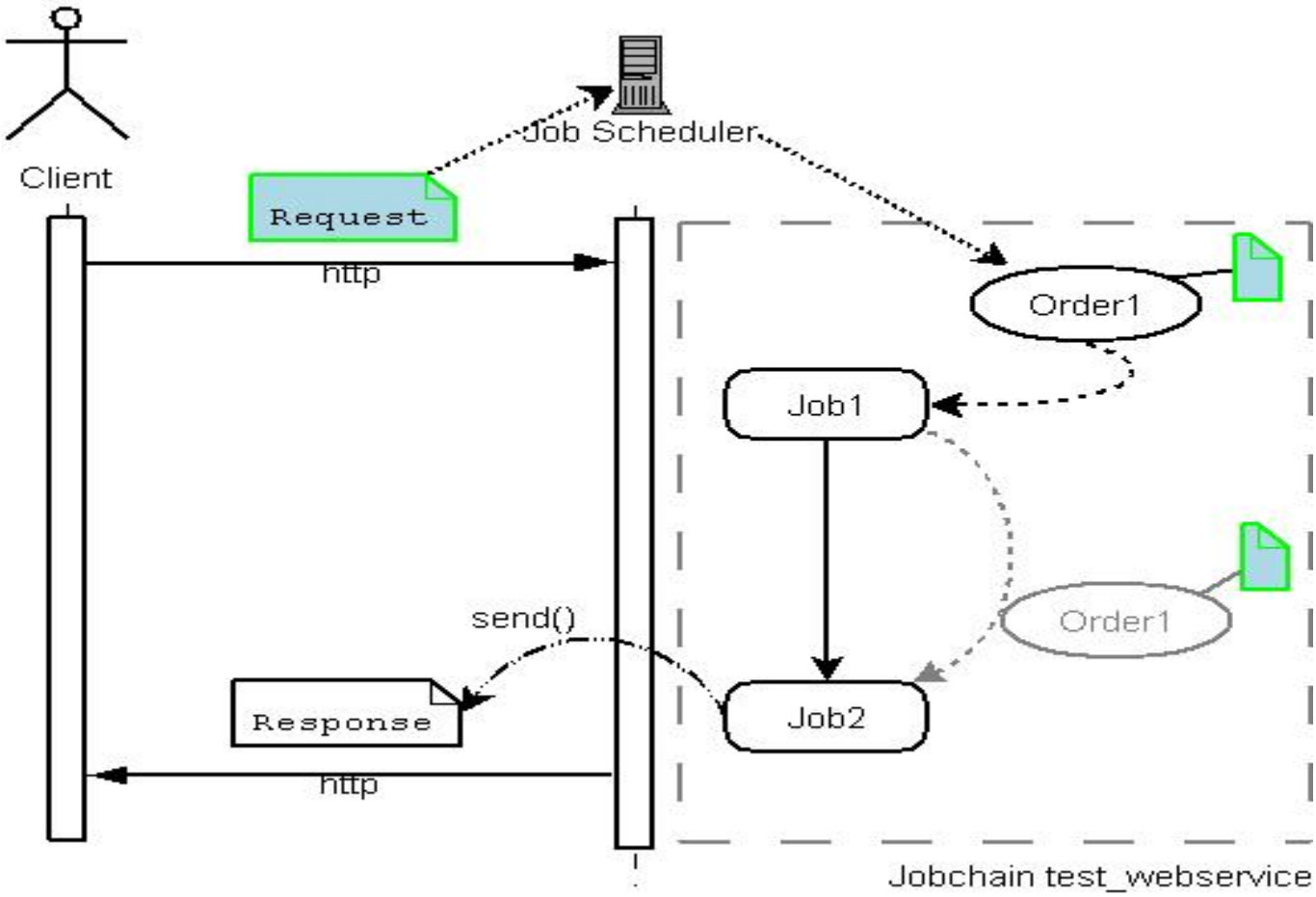
6 The synchronous Web Service

Consider the situation where a Web Service has been configured for the `http://localhost:4444/test_webservice` URL and that this service uses a job chain to synchronously reply to Web Service queries. These queries come to the JobScheduler (to the host and TCP port) from the client as HTTP-POST requests.

The JobScheduler would not attempt to interpret this query, but create an order for the job chain which has been pre-configured for Web Services. This order would then contain the query.

The jobs in this Web Services job chain have the possibility of reading the query out of the order and replying to it. Jobs in this job chain must reply to the query using the `Web_service_response.send()` method.




Client
Job Scheduler
Request
http
Order1
Job1
send()
Order1
Response
Job2
http
Jobchain test_webservice

7 Web Service Control

The JobScheduler distribution contains two style sheets, which enable the Web Service functions of the JobScheduler with SOAP. This Web Service is, however, per default inactive.

```
    <web_service
    debug                       =  "no"
    request_xslt_stylesheet     =  "config/scheduler_soap_request.xslt"
    response_xslt_stylesheet    =  "config/scheduler_soap_response.xslt"
    name                        =  "scheduler"
    url_path                    =  "/scheduler" >
    </web_service>
```

Example: scheduler.xml: Web Service activation

To use the web service of the JobScheduler the tag must be inserted in the scheduler.xml. The best way to customize this is to use JOE.

A Web Service will then be available under the

```
http://scheduler_host:scheduler_port/scheduler
```

URL after the JobScheduler has been restarted.

7.1 Starting a Job

The SOAP `<startJob>` method, which is made available in the `http://www.sos-berlin.com/scheduler` namespace is used to start a job. This method recognizes the following parameters, which are then sent as child elements :

- job: the Jobname
- name: the Task Name (optional)
- after: the number of seconds which should be waited before the task is started (optional)
- at: a time at which the task should be started (optional)

Job parameters can be set by sending as many `<param>` elements as required. Note that `<param>` has `<name>` and `<value>` as child elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
 <startJob xmlns="http://www.sos-berlin.com/scheduler">
   <job>scheduler_check_sanity</job>
   <at>2011-03-16 17:05:00</at>
 </startJob>
  </soapenv:Body>
</soapenv:Envelope>
```

Example: startJob for the scheduler_check_sanity Job

A SOAP document is returned as reply, containing the task id allocated by the JobScheduler.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <sos:taskId xmlns:sos="http://www.sos-berlin.com/scheduler">112491</sos:taskId>
 </soapenv:Body>
</soapenv:Envelope>
```

Example: SOAP Answer to startJob

7.2 Starting a job chain

The SOAP addOrder method is used to create an order. This method recognizes the following parameters, which must be sent as child elements of `<addOrder>`:

- jobchain: The order job chain
- id: the order id (optional)
- title: the order title (optional)

Order parameters can be set by sending as many `<param>` elements as required.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <addOrder xmlns="http://www.sos-berlin.com/scheduler">
            <jobchain>database_reports</jobchain>
            <title>soaptest</title>
            <param>
                <name>command</name>
                <value>SELECT * FROM SCHEDULER_MANAGED_ORDERS</value>
            </param>
            <param>
                <name>database_connection</name>
                <value>scheduler</value>
            </param>
            <param>
                <name>scheduler_order_report_mailto</name>
                <value>email@myhost.com</value>
            </param>
            <param>
                <name>scheduler_order_report_asbody</name>
                <value>1</value>
            </param>
            <param>
                <name>scheduler_order_report_subject</name>
                <value>webservice_test 1</value>
            </param>
        </addOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

Example: addOrder for the database_reports Job Chain (from the Managed Jobs Packet)

A SOAP document containing the id assigned by the JobScheduler to the order is then sent as a reply.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <sos:orderId xmlns:sos="http://www.sos-berlin.com/scheduler">16120</sos:orderId>
  </soapenv:Body>
</soapenv:Envelope>
```

Example: SOAP Reply to addOrder

8 Configuration of an asynchronous Web Service

In order that the JobScheduler can use the style sheets which are described in the next chapter of this documentation, the *scheduler.xml* configuration file must be modified. First of all a `<http_server>` element must be added after the `</process_classes>` element, should it not already exist.

A `<web_service>` element then needs to be added within the `<http_server>` for each Web Service. The configuration of the Web Service for the example used in this documentation is as follows:

```
<web_service
    debug                    = "yes"
    request_xslt_stylesheet  = "config/webservice/execute_file_request.xslt"
    response_xslt_stylesheet = "config/webservice/execute_file_response.xslt"
    forward_xslt_stylesheet  = "config/webservice/execute_file_forward.xslt"
    name                     = "execute_file"
    url_path                 = "/execute_file" />
```

Example: Web Service config

The `debug="yes"` parameter causes the results of the XSL transformation to be saved in the JobScheduler log directory. This is useful for monitoring whether a transformation has been carried out as expected, but should be switched off in a productive system as this can create large amounts of data.

Note that in a real JobScheduler installation, the paths to the directories in which the style sheets are to be found may differ from those given in the example above. Note also that the JobScheduler must be restarted before changes made to the *scheduler.xml* file are implemented.

9 Configuration of a Job Chain as an asynchronous Web Service

In the following example it is assumed that the JobScheduler has been installed together with the Managed Jobs packet. A preconfigured job chain is included in the Managed Jobs packet. This job chain includes the JobSchedulerManagedExecutableJob, which in turn starts executable files on the JobScheduler host.

Note that the Managed Jobs packet is not a requirement for Web Services with the JobScheduler, it is only used for this example.

A SOAP interface should be made available for this job chain which allows executable files to be processed on the JobScheduler host. A UNIX "ls" command, which writes the content of a working directory to stdout, is used as an example call. An asynchronous answer containing the stdout standard output from the executable file should be available as an option.

The complete style sheet source code can be found in the appendices.

9.1 The Style Sheet "request_xslt_stylesheet"

The SOAP query itself forms the starting point for the development of style sheets to accept SOAP queries and transform them into XML commands which the JobScheduler can understand. The SOAP interface uses an `executeFile` method, with a `file` parameter to specify the request.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsa:To xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
      http://localhost:4444/execute_file
    </wsa:To>
  </soapenv:Header>
  <soapenv:Body>
    <executeFile xmlns="http://www.sos-berlin.com/scheduler">
      <file>ls</file>
    </executeFile>
  </soapenv:Body>
</soapenv:Envelope>
```

Example: SOAP Request Method executeFile

Before the JobScheduler transforms the request, it embeds it in a `<service_request>` document, which transports additional meta-information about the query.

```
<?xml version="1.0" encoding="UTF-8"?>
<service_request url="http://localhost:4444/execute_file">
  <web_service name="execute_file"
               url_path="/execute_file"
               request_xslt_stylesheet="config/webservice/execute_file_request.xslt"
               response_xslt_stylesheet="config/webservice/execute_file_response.xslt"
               debug="1"/>
  <content>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
       ...
    </soapenv:Envelope>
  </content>
</service_request>
```

Example: service_request

Note that the style sheet must stem from a `<service_request>` root:

```
<xsl:template match="/service_request">
```

```
  <xsl:apply-templates select="content/soapenv:Envelope/soapenv:Body"/>
</xsl:template>

<xsl:template match="soapenv:Body">
  <xsl:apply-templates select="*" mode="soapBody"/>
</xsl:template>
```

The second template does not only select the expected `executeFile` method, but all elements in the "soapBody" mode. On the one hand, this is done so that differing methods can be handled by a style sheet (and Web Service). In this case a new template with the "soapBody" mode would be written for each method. On the other hand, this allows valid SOAP requests which contain an unknown method to return a SOAP error. This topic will be covered in more detail later. In this example, a template is required for the `<execute_file>` elements.

```
<xsl:template match="sos:executeFile" mode="soapBody" priority="2">
    <add_order job_chain="executable_files">
    <xsl:attribute name="web_service">
      <xsl:value-of select="/service_request/web_service/@name"/>
    </xsl:attribute>
    <params>
        <param name="command">
          <xsl:attribute name="value">
            <xsl:value-of select="sos:file"/>
          </xsl:attribute>
        </param>
    </params>
  </add_order>
</xsl:template>
```

Example: Template for executeFile

The template generates an `<add_order>` command for the job chain executable files. Further, the `web_service` attribute containing the names of the Web Services being called (see Configuration) is added to the `<add_order>` command.

```
<?xml version="1.0" encoding="UTF-8"?>
<add_order job_chain="executable_files" web_service="execute_file">
  <params>
 <param name="command" value="ls"/>
  </params>
</add_order>
```

Example: Order: executable_files

The transfer of the name of the Web Service to the order is a prerequisite for the JobScheduler later returning an asynchronous answer or making a redirection.

The order parameters are set in `<params>`. The JobSchedulerManagedExecutableJob recognizes the `command` parameter which in turn contains the command to be executed (in this case "ls").

A further template is used for the treatment of errors. This template is used for all method calls which have not been matched to other templates. In the example used in this documentation this template would be used for all methods other than `executeFile`.

```
<xsl:template match="*" mode="soapBody" priority="0.5">
   <!-- Generate SOAP error message for the Web Service -->
   <service_response>
  <content>
    <soapenv:Envelope>
   <soapenv:Body>
     <soapenv:Fault>
    <faultcode>SOAP-ENV:Client</faultcode>
    <faultstring>unknown command: <xsl:value-of select="name(.)"/>
    </faultstring>
     </soapenv:Fault>
   </soapenv:Body>
    </soapenv:Envelope>
  </content>
   </service_response>
</xsl:template>
```

Example: treatment of errors

This template generates the JobScheduler <service_response> command. This command then returns the content of the <content> element as an (asynchronous) response and thereby generates a SOAP error message for all unknown method calls.

9.2 The Style Sheet "response_xslt_stylesheet"

After the JobScheduler has taken on an order, it either returns an answer containing the <OK> element as well as other information about the new order or, in the event of an error, the <ERROR> element.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<spooler>
  <answer time="2011-02-14 12:19:46.342">
    <ok>
      <order order="3823" id="3823" state="0" initial_state="0"
             job_chain="executable_files" job="scheduler_managed_executable_file"
             priority="0" created="2011-02-14 12:19:46.342" web_service="execute_file">
        <log level="debug9"/>
      </order>
    </ok>
  </answer>
</spooler>
```

Example: The JobScheduler Response

The response from the JobScheduler is transformed using the response_xslt_stylesheet and synchronously sent back to the client. This requires that a <service_response> command is generated for the JobScheduler:

```
<xsl:template match="/spooler/answer">
  <service_response>
    <content>
      <soapenv:Envelope>
        <soapenv:Body>
          <xsl:apply-templates select="ERROR | ok"/>
        </soapenv:Body>
      </soapenv:Envelope>
    </content>
  </service_response>
</xsl:template>
```

Example: template: spooler/answer

a response such as the ID given to the new order can then be synchronously returned to the client, should the order be accepted:

```
<xsl:template match="ok[order]" priority="2">
  <sos:orderId><xsl:value-of select="order/@order"/></sos:orderId>
</xsl:template>
```

Other (synchronous) responses are also possible - for example, a Boolean value indicating the success of a command.

9.3 The Style Sheet "forward_xslt_stylesheet"

After an order has been completed by the JobScheduler, a result can be delivered to the Web Service or to another address asynchronously. This involves a further XSLT transformation of either an `<order>` or a `<task>` element, depending on whether or not an `add_order` or a `start_job` command has executed by the Web Service.

Information about an order and the order parameters are contained in the `<order>` element. The `std_out_output` parameter containing the STDOUT output from a command is generated by the "JobSchedulerManagedExecutableJob" job after an order has been executed.

This output is then transformed using the style sheet before being forwarded as a SOAP response.

```
<xsl:template match="order">
  <service_request url="http://localhost:5555/webservice">
    <content>
      <soapenv:Envelope>
        <soapenv:Body>
          <xsl:apply-templates select="payload/params/param[@name='std_out_output']"/>
        </soapenv:Body>
      </soapenv:Envelope>
    </content>
  </service_request>
</xsl:template>
```

Example: Template:order

It is necessary that a `<service_request>` element is generated here as well. To do this a `url` parameter needs to be created for the return address, as it is necessary to establish a new connection for the asynchronous reply. In this example, the URL is hard coded in the style sheet. Note that hard coded URLs only make sense when it is planned that responses are always to be returned to this one address.

In practice, it is more usual that the response address is contained in the query from the Web Service client. In this case, the request_xslt_stylesheet must write the return address in an order parameter, which can then be reread by forward_xslt_stylesheet.

The template then selects the parameter named "std_out_output" and writes the content in a new element in the response:

```
<xsl:template match="param[@name='std_out_output']">
  <sos:stdOut>
    <xsl:value-of select="@value"/>
  </sos:stdOut>
</xsl:template>
```

Example: Template get Stdout

In this way the standard output (stdout) can be forwarded to the application receiving the Web Service responses. Note that it is important that the application which processes such output correctly interprets the new lines in the `<sos:stdOut>` element as genuine new lines.

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
 <sos:stdOut xmlns:sos="http://www.sos-berlin.com/scheduler">bin
config
jobs
lib
logs
mail
web
 </sos:stdOut>
  </soapenv:Body>
</soapenv:Envelope>
```

Example: JavaScript®: HelloWorld

10 Configuration of a synchronous Web Service

Before the JobScheduler can use the job described in the following chapter for a Web Service, it is necessary to configure a job chain in the live folder.

The job chain for the example in the next chapter is configured as follows:

```
<job_chain name="web_service_test">
   <job_chain_node state="1" job="web_service_test_job" next_state="100" error_state="999" />
   <job_chain_node state="100"/>
   <job_chain_node state="999"/>
</job_chain>
```

Example: job chain: web_service_test

Although job chains usually consist of a number of jobs, the example chain has only the one `web_service_test_job` job. This job is configured as follows in the live folder:

```
<job name       = "web_service_test_job"
     order      = "yes">
    <script language      = "java"
            java_class    = "sos.scheduler.service.WebServiceTestJob"/>
</job>
```

Example: Job: web_service_test_job

In order that this job chain can be used as a synchronous Web Service, a Web Service definition needs to be created in the `<web_services>` element (this element is described in "Configuration of Asynchronous Web Services"):

```
<web_service name="test"
  job_chain="web_service_test"
  url_path="/test"/>
```

Example: JavaScript®: HelloWorld

Note that the JobScheduler must be restarted after the *scheduler.xml* file has been changed.

11 The Creation of Jobs for Synchronous Web Services

This chapter shows how a job which can read a Web Service request and return a synchronous response could be implemented.

The function of this job is the interpretation of possible protocols such as SOAP. The job receives the request content in either binary or string form. Note that in order to keep this example simple, XML will not be interpreted here, as this would make the code longer and more complicated. The use of the DOM is also not covered by this documentation. Therefore the following job only logs the request content and returns a string containing the task ID.

```
package sos.scheduler.service;

import sos.spooler.Job_impl;
import sos.spooler.Order;
import sos.spooler.Web_service_operation;
import sos.spooler.Web_service_request;
import sos.spooler.Web_service_response;

public class WebServiceTestJob extends Job_impl {

  public boolean spooler_process() throws Exception {
    Order order = spooler_task.order();
    Web_service_operation operation;
    try {
      operation = order.web_service_operation();
    } catch (Exception e){
      spooler_log.info("There is no Web Service operation attached to this order");
      return true;
    }
    // read request
    Web_service_request request = operation.request();
    spooler_log.info("Content of Web Service Request:\n"+request.string_content());

    // send reply
    Web_service_response response = operation.response();

    response.set_string_content("Task ID: "+spooler_task.id());
    response.send();
    return true;
  }
}
```

Example: Implementation: WebServiceTestJob.java

The `web_service_operation()` call is placed in a `try-catch` block at the start of the job. This is necessary as it is possible that the job has to process an order which was created by the JobScheduler but had not been processed before the Scheduler shut down. In this case the JobScheduler would load the order from its database although the HTTP connection to the client no longer existed. Further, the `Web_service_operation` object would no longer exist. The job should catch this error and decide how to handle the order.

The request content is read out of the `Web_service_request` object and written in the log.

The answer is added to the `Web_service_response` object and synchronously sent back to the client using `send()`.

In practice the `Web_service_request` would be read by one job and the `Web_service_response` set by another. It would be possible to create a job chain in which the first job reads the `Web_service_request` object and evaluate and write request parameters into the order payload. Subsequent jobs would then process the order payload and only the last job would write in the `Web_service_response` object and call the `send()` method.

The example job described above is also available for download in the JobScheduler distribution samples package in versions for JavaScript® and Perl. These versions can be found under the following names:

- web_service_test_job.js
- web_service_test_job.pl

In addition to the creation of a job, the example configurations included in the samples package create job chains and Web Services.

Appendix A: The execute_file_request.xslt Style Sheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sos="http://www.sos-berlin.com/scheduler"
     exclude-result-prefixes="soapenv sos" version="1.0">
<xsl:output method="xml" indent="yes"/>

  <xsl:template match="/service_request">
     <xsl:apply-templates select="content/soapenv:Envelope/soapenv:Body"/>
  </xsl:template>

  <xsl:template match="soapenv:Body">
     <xsl:apply-templates select="*" mode="soapBody"/>
  </xsl:template>

  <xsl:template match="*" mode="soapBody" priority="0.5">
     <!-- Generate SOAP Error Message for the Webservice -->
     <service_response>
       <content>
         <soapenv:Envelope>
         <soapenv:Body>
               <soapenv:Fault>
                 <faultcode>SOAP-ENV:Client</faultcode>
                 <faultstring>unknown command: <xsl:value-of select="name(.)"/>
                 </faultstring>
               </soapenv:Fault>
         </soapenv:Body>
      </soapenv:Envelope>
       </content>
     </service_response>
  </xsl:template>

  <xsl:template match="sos:executeFile" mode="soapBody" priority="2">
    <add_order job_chain="executable_files">
      <xsl:attribute name="web_service">
        <xsl:value-of select="/service_request/web_service/@name"/>
      </xsl:attribute>
      <params>
          <param name="command">
            <xsl:attribute name="value">
              <xsl:value-of select="sos:file"/>
            </xsl:attribute>
          </param>
      </params>
    </add_order>
  </xsl:template>

</xsl:stylesheet>
```

Example: execute_file_request.xslt

Appendix B: execute_file_response.xslt

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sos="http://www.sos-berlin.com/scheduler"
     exclude-result-prefixes="soapenv sos" version="1.0">
<xsl:output method="xml" indent="yes"/>

  <xsl:template match="/spooler/answer">
    <service_response>
      <content>
       <soapenv:Envelope>
         <soapenv:Body>
           <xsl:apply-templates select="ERROR | ok"/>
         </soapenv:Body>
       </soapenv:Envelope>
      </content>
    </service_response>
  </xsl:template>

  <xsl:template match="ERROR">
    <soapenv:Fault>
                <faultcode>SOAP-ENV:Client</faultcode>
                <faultstring><xsl:value-of select="@text"/></faultstring>
    </soapenv:Fault>
  </xsl:template>

  <xsl:template match="ok[order]" priority="2">
    <sos:orderId><xsl:value-of select="order/@order"/></sos:orderId>
  </xsl:template>

</xsl:stylesheet>
```

Example: execute_file_response.xslt

Appendix C: execute_file_forward.xslt

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sos="http://www.sos-berlin.com/scheduler"
     exclude-result-prefixes="soapenv sos" version="1.0">
     <xsl:output method="xml" indent="yes"/>

     <xsl:template match="order">
       <service_request url="http://localhost:5555/replyUrl">
         <content>
           <soapenv:Envelope>
         <soapenv:Body>
           <xsl:apply-templates select="payload/params/param[@name='std_out_output']"/>
         </soapenv:Body>
       </soapenv:Envelope>
         </content>
       </service_request>
     </xsl:template>

     <xsl:template match="param[@name='std_out_output']">
       <sos:stdOut>
         <xsl:value-of select="@value"/>
       </sos:stdOut>
     </xsl:template>
</xsl:stylesheet>
```

Example: execute_file_forward.xslt

Index

W