PA/4 WS Tutorial

How to Implement a Migrate Web Service

Version:1.0

To:

Christen Hedegard

PA/4 Work Package Lead

PLANETS Project

Cc:

Amir Bernstein

Swiss Federal Archives

PLANETS Project

July 7, 2009

Author: Dr. sc. math. Hartwig Thomas, Enter AG, Zurich, Switzerland

1 SUMMARY

This „tutorial“ serves the purpose of guiding the development of „Tools for Objects“ as Web Services within the Work Package PA/4 of the sub project Preservation Action (PA) of the EU project PLANETS (Preservation and Long-term Access through NETworked Services: http://www.planets-project.eu/).

Such a tutorial was deemed necessary as both the JAX-WS documentation as well as the documentation of the PLANETS Integrated Framework (IF) leave many questions open.

This tutorial addresses general questions of platform-independence, endorsed libraries, packaging and deployment of services, versioning of Web Services. In addition it contains specific instructions for implementing a Migrate Web Service within PA/4.

This tutorial is designed as a supplement to the IF specification currently available. It goes beyond the current IF status with respect to large objects (streaming), complex objects (zip files) and object references (URIs).

The tutorial leads the reader through a number of steps in developing the generic command-line *Migrate* wrapper and the generic *Migrate* client. It is accompanied by code in a number of project versions which should be consulted while reading this document.

PA/4 WS Tutorial

How to Implement a Migrate Web Service

TABLE OF CONTENTS

2 STATUS OF WEB SERVICE DEVELOPMENT FOR PA/4 TODAY

Up to now the developers within PA/4 have attempted to „wrap“ their Tools for Objects within the given framework established by the sub project Integration Framework (IF) of the PLANETS project.

The experience with a first version of the *Migrate* interface within IF has shown, that it is possible to implement PA/4 services using it. It has, however, also shown, that the such PA/4 services are severely limited in a number of ways:

– The IF framework is not only dependent on the Web Services framewrok employed (*JAX-WS 2.1.5*), but also on an old *JAVA* (*JAVA SE 1.5*) version as well as on the application server (*JBoss*), making development, testing and deployment difficult.

– The *JAX-WS* framework and its *WSIT Tutorial* are heavily dependent on *NetBeans* as a development environment. The IF documentation primarily refers to the *WSIT Tutorial* and thus shares this dependence on a specific IDE belonging to Sun (now „eclipsed“ by Oracle).

– The IF framework does not address the question of versioning. Thus today when the *Migrate* interface was modified recently by IF, most PA/4 services cannot be addressed and run any more.

– The *Migrate* interface together with its *DigitalObject* and dependent data type classes do not permit to migrate „large“ objects. (By „large“ we mean any object that cannot be accommodated within *JAVA* memory at run time. Typically this means that it is impossible to migrate an *MDB* file larger than 30 MB within the IF.)

– The *Migrate* interface does not address the question of complex objects. In the IF/Service Developers Workshop 4 which took place on March 23-25, 2009 in Cologne, this question was addressed and only partly resolved by deciding that „complex objects“ would have to be zipped.

– The *Migrate* interface currently does not support the passing a digital object by reference (e.g. a *JDBC-URI*) rather than by value. This prevents the implementation of database migration from live Oracle- oder SQL-Server databases.

After the workshop in Cologne the *Migrate* interface has been changed by the IF group but has not reached a point of declared stability. Thus many „wrapped“ services of PA/4 cannot be run on today‘s version of the IF.

If the developers of PA/4 want to make any progress they have to take the development of the *Migrate* interface and its services into their own hands. This tutorial is intended to support that effort.

It has been customary within the PLANETS project up to today that developers just communicate with each other by presenting running code. It was felt by the author of this tutorial, that producing a running commentary on the decisions made would be very helpful in a situation where the PLANETS project has immersed itself in the „leading edge“ of Web Service development (the Metro project changing the basis every year!) which is not documented rationally either and turns out to be the „bleeding edge“ for the many productive programmers of the project.

3 PREREQUISITES AND REQUIREMENTS

The *Migrate* services of PA/4 must conform to prerequisites that were decided at the outset of the PLANETS project. In addition we want them to fulfill a number of requirements.

3.1 Prerequisites

The *Migrate* services of PA/4 must – as all services of PLANETS – be available as Web Services. These must be published in the form of a *WSDL* (Web Service Description Language) file determining the contract between the service and its clients.

The *JAVA* development of these services is to be based on the *JAXWS2.1.5-20081030.jar*. The *JAX-WS* jar files are the only addition to a standard *JAVA SE 1.5* or *1.6* installation needed to develop and run the *Migrate* services.

3.2 Requirements

Each *Migrate* service of PA/4 is published as a separate service. The deployment container of each *Migrate* service is a *WAR* (Web ARchive) file.

The *Migrate* services of PA/4 should all implement the „same“ *Migrate* interface, making it possible to call them interchangeably from a PLANETS work flow. (But without different *versions* of the *Migrate* interface there can be no progress!)

The *Migrate* services of PA/4 should not depend on the usage of an old *JAVA* version or on a specific application service. Their usage should impinge minimally on the environment. Particularly careful handling on „endorsed“ *JAR* (Java ARchive) files must ensure that a change in the server‘s environment does not break many other services deployed.

The *Migrate* services of PA/4 should not depend on specifics of a particular IDE.

The *Migrate* services should be subjected to a versioning concept that makes it possible for older versions to coexist besides more recent versions of a service or a service interface.

The *Migrate* services must be able to handle migration of „large“ objects using a streaming interface.

The *Migrate* services must support the passing of a digital object „by reference“ (e.g. through a *JDBC URI*) as well as „by value“ (as a byte stream).

In order to enable progress – within IF, within PLANETS –, separate versions of the *Migrate* interface must be supported. It is to be expected that not all PA/4 services implement the same version of the *Migrate* interface at any time. Thus any work flow making use of the PA/4 services must be ready to handle more than one version of the *Migrate* interface. In particular it needs to support precursors of the „current“ version.

4 INSTALLING JAX-WS

The *JAXWS2.1.5-20081030.jar* can be downloaded from here: https://jax-ws.dev.java.net/2.1.5/. It can be unzipped to any location. Its documentation and samples are useful, though sometimes cryptic and sometimes frustratingly avoiding answering everyday questions.

For development we only need the 19(!) *JAR* files from its *lib* directory which we will collectively call the *jaxws-ri* JARs.

These *JARs* are needed for the client as well as for the service. In order to enable a Web Container like *Tomcat* or *JBoss* for this version of Web Services the *jaxws-ri JARs* need to be copied to some shared library folder of the application server, where they are picked up by its class loaders at startup time.

Particularly the two API *JARs jaxws-api.jar* and *jaxb-api.jar* need to be „endorsed“ by the container. This is necessary because JAVA SE 1.6 distributes an older version (2.0) of JAX-WS as part of its default run-time libraries. The „endorsed“ mechanism tells the class loaders to prefer the more recent JAX-WS 2.1.5 version over the JAVA SE 1.6 run-time libraries.

This „JAR war“ has become more annoying than the „DLL hell“ ever was! It is not recommended to run the *Ant* task of the *JAX-WS* distribution for installing it, because that will attempt to install the endorsed *JARs* to your JDK installation, thus influencing – and possibly breaking – every other *JAVA* program you run on your machine and preventing you from upgrading to a new version of *JAVA*. More about the „endorsed“ problem later in the context of deployment.

5 A MINIMAL WEB SERVICE

5.1 Directory Structure

Because we want to implement a Web Service we need the following directory structure:

Generic-0-0	the project directory (version 0.0, because it is very minimal)
build	the JAVA classes (build target)
src	the JAVA source of the Web Service
webapp	the web content
jaxws-ri	the unzipped contents of JAXWS2.1.5-20081030.jar (outside the project)

5.2 Interface (SEI)

We start with a JAVA interface which will embody the service contract in the *WSDL*. Its package name will be *eu.planets_project.services.migration*. Its name will be *Migrate_0.java*. We put it into the corresponding folders under *src*. It is a good practice to separate the JAVA interface (corresponding to a *WSDL* „Port“) from the JAVA implementation (corresponding to a *WSDL* Service). Often[1] the interface is called *SEI* (Service Endpoint Interface) and the implementation *SIB* (Service Implementation Bean).

```
/*== Migrate_0.java ==================================================
Minimal PLANETS migration service.
Version     : $Id: Migrate_0.java 126 2009-07-17 14:26:13Z hartwig $
Application : PLANETS PA/4 migration services
Description : Migrate_0 defines a minimum interface for migration
Platform    : JAVA SE 1.5 or higher, JAX-WS2.1.5 (2008/10/30)
----------------------------------------------------------------------
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 07, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
====================================================================*/
package eu.planets_project.services.migrate;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebParam;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

/*==================================================================*/
/** This interface corresponds to the service contract which is
 * represented by WSDL which is automatically generated by the annotation
 * processing of JAX-WS 2.1.5.
 * @author Hartwig Thomas <hartwig.thomas@enterag.ch>
 */
@WebService(targetNamespace=Migrate_0.sNS, name=Migrate_0.sNAME+"-"+Migrate_0.sVERSION)
public interface Migrate_0
```

[1]e.g. in the book *Java Web Services – Up and Running* by Martin Kalin, O'Reilly 2009.

```
{
  String sNS = "http://migrate.services.planets_project.eu/";
  String sNAME = "Migrate";
  String sVERSION = "0";

  /*------------------------------------------------------------------*/
  /** transform the original (String) object and return the migrated object.
   * @param sOriginal the original (String) object.
   * @return the migrated (String) object.
   */
  @WebMethod
  @RequestWrapper(className="MigrateRequest_"+Migrate_0.sVERSION)
  @ResponseWrapper(className="MigrateResponse_"+Migrate_0.sVERSION)
  @WebResult(name="sMigrated") String migrate(
    @WebParam(name="sOriginal") String sOriginal);

} /* interface Migrate_0 */
```

The actual code of this interface is minimal. We encourage documenting the code systematically and supporting the *Javadoc* annotations. After all this is a service contract and thus deserves to be documented clearly so that developers implementing clients can understand what is happening.

The strings for name space, name and version will be needed – and explained – later.

We have chosen a middle road for annotating the interface. We could have dispensed with the *WebMethod, WebParam*, *RequestWrapper* and the *ResponseWrapper* annotations. Choosing those names explicitly help, however, to find the corresponding elements in the generated *WSDL*. Also these explicit choices have the effect that no generated artifacts will be needed by the Web Client. Finally, making sure that we explicitly name all *XML* elements has the effect, that different application servers are garanteed to dynamically generate the same XML element names in the interface.

The name in the *WebService* annotation chosen here, will appear as the „Port“ name in the generated *WSDL* document.

Hyphens and Underscores

It may be noted, that the project name and the PLANETS domain name contains hyphens, whereas the very similar class name and package name contain underscores. This is due to the fact, that *JAVA* identifiers cannot contain hyphens. We stick to the convention, that we prefer hyphens in URLs and use underscores in corresponding *JAVA* identifiers.

5.3 Implementation (SIB)

Next we implement a JAVA class which implements the interface and will represent the actual Web Service. We call this class *GenericMigrate_0_0*.

```
package eu.planets_project.services.migration.generic;

import javax.jws.WebService;
import eu.planets_project.services.migrate.Migrate_0;
```

```
/*====================================================================*/
@WebService(
  endpointInterface = "eu.planets_project.services.migrate.Migrate_0",
  portName=Migrate_0.sNAME+"-"+Migrate_0.sVERSION,
  targetNamespace=GenericMigrate_0_0.sNS,
       serviceName=GenericMigrate_0_0.sNAME+"-" + GenericMigrate_0_0.sVERSION)
public class GenericMigrate_0_0 implements Migrate_0
{
  public static final String sNS =
    "http://generic.migration.services.planets_project.eu/";
  public static final String sNAME = "GenericMigrate";
  public static final String sVERSION = Migrate_0.sVERSION + "-0";

  /*------------------------------------------------------------------*/
  /* (non-Javadoc)
   * @see eu.planets_project.services.migrate.Migrate_0#migrate(java.lang.String)
   * migrate just implements an "echo" service.
   */
  public String migrate(String sOriginal)
  {
    return sOriginal;
  } /* migrate */

} /* GenericMigrate_0_0 */
```

Here we have removed some lengthy comments. The important annotation is the *endpointInterface* attribute in the *WebService* annotation. The name and version of the implementation will become more clear later. Here it suffices to say, that we expect the interface (and thus the *WSDL* document) to change less often than the implementation. It must be noted, however, that a change in the implementation (semantics) even with a stable interface (syntax) will break „old“ clients, which most likely will rely on the semantics just as much as on the syntax.

The name in the *WebService* annotation chosen here will appear as the „Service“ name in the generated *WSDL* document.

5.4 Publisher

A nice feature of *JAX-WS* is the fact, that the Web Service can be published outside of the context of an application server like *Glassfish*, *Tomcat* or *JBoss*. This is particularly interesting in view of the „endorsed“ problem discussed in more detail below. It is also very useful for testing during the development process and, of course, very useful for demonstration purposes in an „I carry it all on my notebook“ situation.

„Publishing“ means establishing a concrete listening connection to a specific TCP/IP port under a specific host name, which can be addressed by a client of the the Web Service. The parameters of the Publisher are therefore the host name and the TCP/IP port. This is a port number and not to be confused with the WSDL „Port“ name which refers to the abstract interface implemented by the service.

```
package eu.planets_project.services.migration.generic;

import javax.xml.ws.Endpoint;


/*====================================================================*/
public class GenericMigratePublish_0_0
{
  private static final String sDEFAULT_HOST = "localhost";
  private static final String sDEFAULT_PORT = "8080";
  private static final String sSERVICE_PATH = "GenericMigrate-0-0";


  /*-------------------------------------------------------------------*/
  /** display usage information
   */
  private static void displayHelp()
  {
    System.out.println("Usage:");
    System.out.println("java " +
      "-Djava.endorsed.dirs=../lib/endorsed -cp <classpath>\n" +
      "-Djava.util.logging.config.file=\"../etc/logging.properties\"\n" +
      "eu.planets_project.services.migrate.GenericMigratePublish [<host>]");
    System.out.println("with");
    System.out.println("<classpath> must point to the class to be executed");
    System.out.println("            (e.g. build/classes");
    System.out.println("<host>      host where Web Service is to be published");
    System.out.println("            default: localhost:8080");
  } /* displayHelp */

  /*-------------------------------------------------------------------*/
  /** main expects host paramter where service is to be published.
   * @param args none for default (localhost:8080), -h for help, or
   *             host or hst:port
   */
  public static void main(String[] args)
  {
    if ((args.length > 0) && (args[0] == "-h"))
      displayHelp();
    else
    {
      try
      {
        String sUrl = sDEFAULT_HOST+":"+sDEFAULT_PORT;
        if (args.length > 0)
          sUrl = args[0];
        if (!sUrl.startsWith("http://"))
          sUrl = "http://" + sUrl;
        if (sUrl.endsWith("/"))
          sUrl = sUrl.substring(0,sUrl.length()-1);
        if (sUrl.indexOf(':') < 0)
          sUrl = sUrl+":"+sDEFAULT_PORT;
        sUrl = sUrl + "/" + sSERVICE_PATH;
        System.out.println("Service will be available under "+sUrl);
        System.out.println("User Ctrl-C to stop the service.");
        /* now publish */
        Endpoint.publish(sUrl,new GenericMigrate_0_0());
      }
      catch (Exception e)
      { System.err.println(e.getClass().getName()+": "+e.getMessage()); } }
  } /* main */

} /* class GenericMigratePublish_0_0 */
```

The *displayHelp* method reminds us, that we have to be careful about the „endorsed“ problem.

This trivial version of the Publisher implements a single-threaded service which is not quite suitable for „production“. For development testing, however, it is very convenient, due to its short start-up delay (~1s as opposed to *Tomcat* 4 s, or *JBoss* 300 s).

We chose to append the path *GenericMigrate-0-0* to the host name for compatibility with the way application servers tend to start the path of an application with the application‘s name.

As a convenience for the user we display the *URL*, where the service will be available.

It is worth noting, that this is the only place in the code where the TCP/IP port number appears. When we address the question of publishing the Web Service to an application server we will see, that only the configuration of the application server determines the host name and the port number under which the Web Service is published. In the „abstract“ code of the Web Service these concrete bindings appear nowhere – thus enabling us to publish it to any URL.

5.5 Building the Service

Now we can compile those three classes. Rather than doing this from the command line, we add the following *build.xml* to the project folder and execute it using *Ant*. Using an *Ant* task for the build keeps things reproducible and permits us to add tasks later.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project basedir="." default="build" name="GenericMigrate-0-0">
  <property name="debuglevel" value="source,lines,vars"/>
  <property name="target" value="1.5"/>
  <property name="source" value="1.5"/>

  <target name="init">
    <mkdir dir="build/classes"/>
    <copy includeemptydirs="false" todir="build/classes">
      <fileset dir="src" excludes="**/*.launch, **/*.java"/>
    </copy>
  </target>

  <target name="clean">
    <delete dir="build/classes"/>
  </target>

  <target depends="init" name="build">
    <echo message="${ant.project.name}: ${ant.file}"/>
    <javac debug="true" debuglevel="${debuglevel}"
      destdir="build/classes"
      source="${source}"
      target="${target}">
      <src path="src"/>
    </javac>
  </target>

</project>
```

We chose „1.5“ for the source and target JAVA version. That is the first version which supports annotation. We do not depend on any more recent features of *JAVA*. So we might as well make it accessible to the largest number of users.

Annotation processing

JAX-WS works a lot with *JAVA*-annotations. We use *JAX-WS* in a fashion that does not make use of artifacts generated by the annotation processor. For other approaches to Web Services such artifacts, however, are needed. In an *Ant* build script one would likely use a call to *wsgen* in the *bin* folder of the *JAX-WS* distribution in order to generate those artifacts. In *NetBeans JAX-WS* is tightly integrated – thus no additional activity is needed. An equally convenient integration into *Eclipse* can be achieved this way: In the properties of the project choose Java *Compiler > Annotation Processing*. There enable the project specific settings and then continue to *Factory Path*. There click *Add External JARs ...* and add *../jaxws-ri/lib/jaxws-tools.jar* as the *JAR* to be used for annotation processing. Finally select this new entry and click *Advanced ...*. There you find that only one annotation processor was found in the *JAR*: *com.sun.istack.ws.AnnotationProcessorFactory*. Select the option *Run this container's processors in batch mode*. Now on each build the annotation processor will generate the necessary artifacts in the project folder *.apt_generated* or in a *jaxws* folder under the class's source folder and the build will compile the generated classes there.

5.6 Running the service

In the folder *build/classes* we can now run the service by executing the following single-line command:

```
java -Djava.endorsed.dirs=../jaxws-ri/lib -cp build/classes
  eu.planets_project.services.migration.GenericMigratePublish_0_0
```

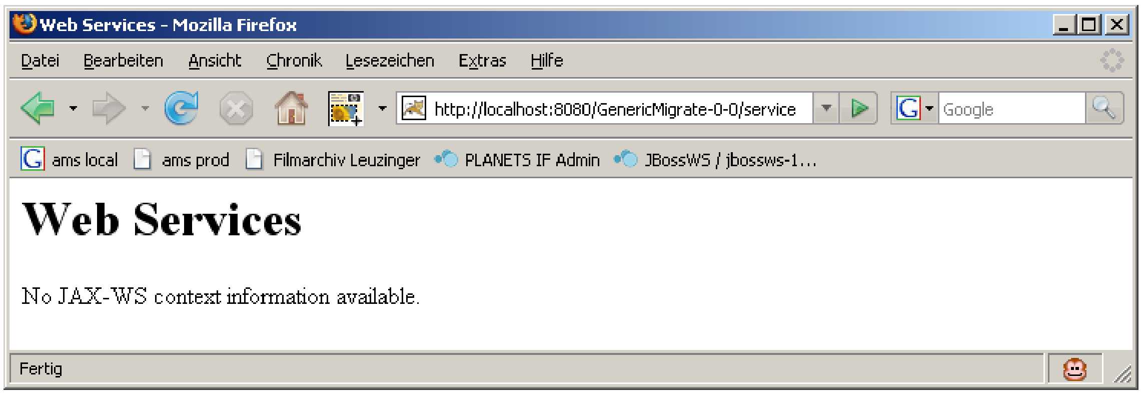
It is not necessary to put the *jaxws-ri JARs* onto the class path as they have already been included in the boot loader's class path using the endorsed mechanism.

The command tells us on *stdout* that the service will be available under *http://localhost:8080/GenericMigrate-0-0*. It then proceeds to write a few info log messages about dynamically created artifacts to *stderr* and starts listening at the port *8080* of the *localhost*. We can stop it with *Ctrl-C*.

5.7 Examining the service

While the service is running, we use the URL *http://localhost:8080/GenericMigrate-0-0* in a browser to examine the service.

The displayed answer tells us that no *JAX-WS* context information is available


Web Services - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost:8080/GenericMigrate-0-0/service
Google
ams local
ams prod
Filmarchiv Leuzinger
PLANETS IF Admin
JBossWS / jbossws-1...
Fertig

Web Services

No JAX-WS context information available.

thus proving, that there *is* a *JAX-WS* endpoint listening at this address.

Entering the same URL with „?wsdl“ appended results in a display of the generated *WSDL* document. This is an already quite complex *XML* file describing the service using almost all the information we put into its annotations.

Appending the query „?xsd=1“ lets us examine the generated XML Schema Definition (*XSD*) which informs us about the structure of data types.

5.8 A Dynamic Client

A Web Service is not mainly to be examined with the help of a browser. Instead, a client application can make use of it, by „calling“ its methods similar to the way any other method is called locally.

The *JAX-WS* environment will ship the parameters across the network to the service. The service will compute the results and the *JAX-WS* environment will return them – again transmitting them through the network – to the caller. This basic mechanism is known as *RPC* (Remote Procedure Call). In the past it has been used as *Sun-RPC* for the Network File System (*NFS*), as *DCE-RPC* in Common Object Request Broker Architecture (*CORBA*) and in Microsoft‘s Distributed Component Object Model (*DCOM*), as Remote Method Invocation (*RMI*) in *JAVA*, as *XML-RPC* and *JAX-RPC* in early versions of „Web Services“. Although we shall use the *DOCUMENT* style of *JAX-WS* Web Services rather than the *RPC* style this only refers to the way the parameters are packaged. From a broader point of view both as *RPC* mechanisms.

There are two ways of creating a client: static and dynamic clients. For a static client one can generate *JAVA* stub classes from the *WSDL* document using the tool *wsimport*. Those classes can be called by the client code and will handle all the marshaling and transmitting of the request and the response.

Alternatively all this can be done dynamically without the need of compiling static classes. (With *JAVA*‘s reflection mechanism any class that can be generated statically can be generated dynamically.) This has the advantage that we can create a rather generic client for all services – including future services for which there exists no *WSDL* yet – implementing the same interface.

Most of the client code is concerned with parameter parsing and niceties for the user. For the developer only the method *getMigrateProxy* is of interest:

```
/*-------------------------------------------------------------------*/
/** getMigrateProxy returns a proxy class implementing the Migrate_0
 * interface and representing the Web Service.
 * @param sWsdlUrl URL of WSDL.
 * @param sService service name.
 * @return output text.
 */
private Migrate_0 getMigrateProxy(String sWsdlUrl, String sService)
  throws MalformedURLException
{
  URL urlWsdl = new URL(sWsdlUrl);
  Migrate_0 migProxy = null;
  /* service */
  Service service =
    Service.create(urlWsdl, new QName(GenericMigrate_0_0.sNS, sService));
  /* port corresponds to the interface */
  migProxy = (Migrate_0)service.getPort(Migrate_0.class);
  System.out.printf("Created proxy class for service %s\n",service.getServiceName());
  return migProxy;
} /* getMigrateProxy */
```

This returns a proxy class for the *Migrate_0* interface which will have the methods of the interface executed by the service.

Executing this client while the publisher is running displays again a number of informatory log messages of the JAX-WS framework on *stderr* and the echoes the input string – as expected.

5.9 What has been achieved so far

We have been able to implement a minimal Web Service which is independent of the *JAVA* version (1.5 or greater), does not contain any *EJB* JARs nor any *JBoss* JARs. It can be run independent of any application server. It can even be productively published without such a „container“. As soon as real security handling is required such a container becomes indispensible, however.

6 ENDORSED JARS, PACKAGING AND DEPLOYMENT

Unfortunately most books about Web Services omit the step of packaging going directly from development to deployment. In practice the packaging step is very important. It gathers all binaries together and hands them to the user (e.g. Web Administrator). Usually it is not the developer who has direct access to the application server where the Web Service is to be deployed. Therefore the central result of development must be packaging rather than deployment.

6.1 Endorsed JARs

Packaging a *JAX-WS* Web Service is complicated by the problem of „endorsed“ *JARs*. The security mechanisms of *JAVA* do not permit the user‘s overdefining of basic *JAVA* run-time classes. This protects all *JAVA* applications from being fooled by a rogue application that redirects their calls to itself. This protection becomes problematic, when the development of new versions of future *JAVA* run-time classes runs ahead of the current *JAVA* distribution as is the case with the *Metro* project developing the *JAX-WS* framework. In order to run the „leading edge“ *JAX-WS* environment we have to override the older standard *JAX-WS* classes in the *JAVA* run-time. (This is mainly a problem when *JAVA SE 1.6* is used as *JAVA SE 1.5* did not include *JAX-WS* classes at all yet.) The developers of the IF project of PLANETS have rightly decided, that PLANETS needs a more recent version of *JAX-WS* than the one distributed with *JAVA SE 1.6*. Otherwise streaming of large objects (see below) would not be possible. (The next version of *JAX-WS 2.1.7* dating from April 2009 is apparently integrated in the most recent version of *JAVA SE 1.6u4*. It seems inadvisable, however, to change *JAX-WS* versions in the middle of the PLANETS project and making usage of this most recent JAVA version mandatory.)

For such cases the mechanism of „endorsed“ *JARs* was introduced into *JAVA*. The basic idea was: If we copy the more recent *JAX-WS JARs* to a specific „endorsed“ folder and make it known to the boot loader, then these classes can be loaded by it instead of the *JAVA* run-time classes with the same name.

We have the following two possibilities to make a JAR into an „endorsed“ *JAR*: Either we create an *endorsed* folder under the *lib* folder of the *jre* of the *JAVA* engine we use, or we can specify any folder containing only endorsed *JARs* with the command-line switch *-Djava.endorsed.dirs=<folder with endorsed jars>* to the *java* executable running our JAVA programs.

The first of these possibilities is not to be recommended because it has side effects on every *JAVA* application run with the currently installed *JAVA* instance thus introducing the risk of breaking such other applications which rely on the run-time they have been developed and tested with. Therefore it is not a good idea to use the *Ant* scripts for installing *JAX-WS* which do precisely this.

The second of these possibilities is used by all application servers. *Glassfish*, *Tomcat*, *JBoss* all have some „endorsed“ folder which is mentioned with the *-Djava.endorsed.dirs* switch on the command line starting the server. The command-line switch possibility can also be used when we start our own publisher and client calls. Our packaging should make it easy for the user to handle the endorsed *JARs*. (For *JAVA 1.6 SE* and *JAX-WS 2.1.5* strictly only the two JARs *jaxb-api.jar* and *jaxws-api.jar* need to be endorsed.

In the case an application server is used it must be noted, that the same problem of possibly breaking other Web Services that rely on the older versions of the *JAX-WS* framework applies. Therefore it is necessary, that a specific *JAX-WS 2.1.5* instance the application server be kept separate from the instance all non-*JAX-WS 2.1.5* services are run on. This specific *JAX-WS 2.1.5* instance needs to be prepared for each application server. As an example we describe the preparation for the popular and fast developer‘s application server *Tomcat 6* and for the very slow and badly documented application server *JBoss 5.1 GA* which was adopted by the IF sub project of the PLANETS project as their deployment server. I am confident that *WebSphere* and other application servers can be similarly prepared.

Tomcat 6

In order to run our *JAX-WS 2.1.5* Web Services on *Tomcat 6* we must copy *jaxws-api.jar* and *jaxb-api.jar* to *%CATALINA_HOME%\endorsed*. All the other *jaxws-ri* JARs can be copied to the *shared/lib* folder of the *Tomcat* instance. They are needed by the Tomcat server to help in anntotation processing at deployment before our own service is running.

JBoss 5.1 GA

It is important, *not* to follow the installation instruction of the *JBoss* documentation for *JAVA SE 1.6*. Instead we just copy *jaxws-api.jar* and *jaxb-api.jar* to *%JBOSS_HOME%\lib\endorsed*. All the other *jaxws-ri JARs* must be copied to *%JBOSS_HOME%\common\lib*. They are needed by the *JBoss* server to help in anntotation processing at deployment before our own service is running.

6.2 Packaging

After having clarified the endorsed problem, we return to the question of packaging. For web applications the standard package format is a *WAR* file. This includes the *JAVA* classes as well as a *WEB-INF* folder with a deployment descriptor *web,xml* and all user *JARs* used by the service.

For deployment we just drop this *WAR* file into a convenient deployment folder of the application server. Most servers support „hot“ deployment, noticing the new *WAR* file, stopping the old version of the same service, if there was one running, unzipping the *WAR* file and deploying the new instance automatically.

For deployment on an application server that has been prepared to support *JAX-WS 2.1.5* we need to package our Web Service as a *WAR* file (e.g. *GenericMigrate-0-0.war*). For deployment on *Tomcat 6* we just copy it to *%CATALINA_HOME%\webapps*. On *JBoss 5.1 GA* we copy it to the folder *%JBOSS_HOME%\server\default\deploy*.

In the case, where we want to run the Web Service without an application server, we just need to distribute our *JAVA* classes. As they depend heavily on *JAX-WS 2.1.5* we should also put the *jaxws-ri* JARs in our package. In addition we must create shell scripts for running the service and the client, which handle the endorsed problem. Finally we should add test data files and documentation (readme, installation instructions, release notes, user‘s manual) to the package.

A useful container for our *JAVA* classes – particularly if they will be increasing in number – is a *JAR* file. This *JAR* file can also be added to the *WEB-INF/lib* folder of the *WAR* file for application servers.

For the stand-alone distribution we shall have to package everything into a *ZIP* file with a *bin* folder holding the shell scripts, a *jaxws-ri* folder holding the *jaxws-ri JARs*, possibly a *doc* folder, holding the *Javadoc* documentation, and a *lib* folder, containing the Web Service *JAR* and – possibly later – other *JARs* needed by the Web Service.

Users of the stand-alone distribution can just run service and client without having their system „polluted“ by endorsed *JARs*. Users of the *WAR* file will have to prepare their application server instance for *JAX-WS 2.1.5* as described above.

6.3 Creating the JAR

The JAR is created, by adding a „jar“ target to the *build.xml*. In it we copy all classes in the folder *build/classes* to the JAR *packages/GenericMigrate_0_0.jar*. In addition we use the file *MANIFEST.MF* under *webapp/META-INF* as the JARs manifest.

```
<target depends="build" name="jar">
  <echo message="${ant.project.name}: jar"/>
  <jar jarfile="${dirbuildlib}/${ant.project.name}.jar"
     manifest="${filemanifest}">
    <fileset dir="${dirclasses}">
      <include name="**/*.class"/>
     </fileset>
    <fileset dir="${dirsrc}">
      <include name="**/*.java"/>
     </fileset>
  </jar>
</target>
```

We have changed the build task such that it copies the sources to the build folder too. In a rather open project like PLANETS it is user-friendly if the sources are included in the distribution.

MANIFEST.MF

For the stand-alone version we choose the service publisher as the main class and add a class path that lists all *JARs* (except for the two „endorsed“ ones) of the *jaxws-ri* distribution that we intend to distribute in the ZIP file of the stand-alone package.

```
Manifest-Version: 1.0
Created-By: Hartwig Thomas for Swiss Federal Archives, Berne, Switzerland
Specification-Title: Migrate
Specification-Version: 0
Specification-Vendor: PLANETS project
ImplementationTitle: GenericMigrate
ImplementationVersion: 0.0
Implementation-Vendor: PLANETS project
Main-Class: eu.planets_project.services.migration.GenericMigratePublish_0_0
Class-Path: ../jaxws-ri/lib/activation.jar
  ../jaxws-ri/lib/FastInfoset.jar
  ../jaxws-ri/lib/http.jar
  ../jaxws-ri/lib/jaxb-impl.jar
  ../jaxws-ri/lib/jaxb-xjc.jar
  ../jaxws-ri/lib/jaxws-rt.jar
  ../jaxws-ri/lib/jaxws-tools.jar
  ../jaxws-ri/lib/jsr173_api.jar
  ../jaxws-ri/lib/jsr181-api.jar
  ../jaxws-ri/lib/jsr250-api.jar
  ../jaxws-ri/lib/mimepull.jar
  ../jaxws-ri/lib/resolver.jar
  ../jaxws-ri/lib/saaj-api.jar
  ../jaxws-ri/lib/saaj-impl.jar
  ../jaxws-ri/lib/stax-ex.jar
  ../jaxws-ri/lib/streambuffer.jar
  ../jaxws-ri/lib/woodstox.jar
Sealed: false
```

6.4 Creating the ZIP

We wish to include the following directories in the binary distribution ZIP file:

– *bin* with *publish.cmd* and *client.cmd* (and later also *publish.sh* and *client.sh*)

– *doc* with Javadoc

– *etc* with *logging.properties*

– *lib/endorsed* with *jaxb-api.jar* and *jaxws-api.jar*

– *jaxws-ri/lib* with all jaxws-ri JARs

– *lib* with GenericMigrate-0-0.jar

Later we will have to add some more documentation in the root folder and support the LINUX platform more fully by adding the corresponding shell scripts. For the moment we have to create the *cmd* files and the *readme.txt*. The command script *publish.cmd* and *client.cmd* include a reference to the *logging.properties* with log level WARNING thus getting rid of unwanted *JAX-WS* messages, but allowing the user to change the log level to *INFO* again.

6.5 Creating the WAR

The appropriate packaging format of a web application for a web container is a *WAR* file. Such a *WAR* file contains a folder *META-INF* with the file *MANIFEST.MF* and a folder *WEB-INF* with a deployment descriptor file *web.xml* and the service‘s classes in the *classes* folder and its *JARs* in the *lib* folder. The *MANIFEST.MF* of the *WAR* file is a shorter version of the *MANIFEST.MF* of the *WAR* file without the *Class-Path* and the *MainClass* entries.

WEB-INF

Before we run the Web Service on an application server we must make it into a valid Web Application. That means that a file *web.xml* in the folder *WEB-INF* must be present.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>GenericMigrateApplication-0-0</display-name>
  <description>Generic Command-line Wrapper Application 0.0</description>
  <listener>
                                                                   <listener-
class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>GenericMigrate-0-0</servlet-name>
    <display-name>GenericMigrateServlet-0-0</display-name>
    <description>Generic Command-line Wrapper Servlet 0.0</description>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>GenericMigrate-0-0</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

From the point of view of the application server our Web Service is a *Servlet* which is implemented by the class *com.sun.xml.ws.transport.http.-servlet.WSServlet* (with a supporting context listener *com.sun.xm-l.ws.transport.http.servlet.WSServletContextListener*). This servlet in turn generates all kind of artifacts (*WSDL*, *XSD*, ...) from the annotations in our code and makes use of the file *sun-jaxws.xml* in order to find our implementation class to be launched.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
  <endpoint
    name='GenericMigrateWebService-0-0'
    implementation='eu.planets_project.services.migration.GenericMigrate_0_0'
    url-pattern='/*'/>
</endpoints>
```

The URL pattern of this file must be identical with the servlet URL pattern in *web.xml*. Choosing „/*“ means, that the application server publishes the service under the name of the WAR file (e.g. *http://localhost:8080/GenericMigrate-0-0*).

WEB-INF must also contain our Web Service classes which are to be launched. As we already have them in a *JAR* file, we store that file in the *lib* folder under *WEB-INF*.

6.6 Full Build Script

The whole packaging is achieved as described by using the following build script:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- $Workfile: build.xml $ ==========================================
ANT build file of GenericMigrate-0-0.
Version    : $Id: build.xml 68 2009-07-12 14:33:27Z hartwig $
Application : PLANETS PA/4 migration services
Description: build file of GenericMigrate-0-0.
Platform   : JAVA, ANT
======================================================================
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 07, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
================================================================== -->
<project basedir="." default="build" name="GenericMigrate-0-0">
  <property name="debuglevel" value="source,lines,vars"/>
  <property name="target" value="1.5"/>
  <property name="source" value="1.5"/>
  <property name="dirsrc" value="src"/>
  <property name="dirbin" value="bin"/>
  <property name="diretc" value="etc"/>
  <property name="dirdoc" value="doc"/>
  <property name="dirwebinf" value="webapp/WEB-INF"/>
  <property name="dirbuild" value="build"/>
  <property name="dirclasses" value="${dirbuild}/classes"/>
  <property name="dirbuildlib" value="${dirbuild}/lib"/>
  <property name="dirbuildendorsed" value="${dirbuildlib}/endorsed"/>
  <property name="dirpackages" value="packages"/>
  <property name="dirjaxws" value="../jaxws-ri"/>
  <property name="dirjaxlib" value="${dirjaxws}/lib"/>
  <property name="filejaxbapi" value="jaxb-api.jar"/>
  <property name="filejaxwsapi" value="jaxws-api.jar"/>
  <property name="filemanifest" value="META-INF/MANIFEST.MF"/>
  <property name="filewebmanifest" value="webapp/META-INF/MANIFEST.MF"/>
  <property name="filewebxml" value="web.xml"/>
  <property name="filesunjaxws" value="sun-jaxws.xml"/>

  <target name="init">
    <echo message="${ant.project.name}: init"/>
    <mkdir dir="${dirdoc}"/>
    <mkdir dir="${dirclasses}"/>
    <mkdir dir="${dirbuildendorsed}"/>
    <mkdir dir="${dirpackages}"/>
  </target>

  <target name="clean">
    <echo message="${ant.project.name}: clean"/>
    <delete dir="${dirdoc}"/>
    <delete dir="${dirbuild}"/>
    <delete dir="${dirpackages}"/>
  </target>
  <target depends="init" name="build">
    <echo message="${ant.project.name}: build"/>
    <javac debug="true" debuglevel="${debuglevel}"
      source="${source}"
      target="${target}"
      srcdir="${dirsrc}"
      destdir="${dirclasses}"
      includes="**/*.java"/>
    <!-- endorsed jars to lib/endorsed -->
    <copy todir="${dirbuildendorsed}">
      <fileset dir="${dirjaxlib}">
        <include name="${filejaxbapi}"/>
        <include name="${filejaxwsapi}"/>
      </fileset>
    </copy>
    <!-- jaxws-ri/lib/* -->
    <copy todir="${dirbuildlib}">
      <fileset dir="${dirjaxlib}">
        <include name="*.jar"/>
      </fileset>
    </copy>
  </target>

  <target depends="build" name="jar">
    <echo message="${ant.project.name}: jar"/>
    <jar jarfile="${dirbuildlib}/${ant.project.name}.jar"
       manifest="${filemanifest}">
      <fileset dir="${dirclasses}">
        <include name="**/*.class"/>
       </fileset>
      <fileset dir="${dirsrc}">
        <include name="**/*.java"/>
       </fileset>
    </jar>
  </target>

  <target depends="init" name="doc">
    <echo message="${ant.project.name}: doc"/>
    <javadoc destdir="${dirdoc}" windowtitle="PLANETS Services Documentation">
      <package name="eu.planets_project.services.*"/>
      <sourcepath location="${dirsrc}"/>
      <bottom>
        <![CDATA[PLANETS Project, Preservation Action]]>
      </bottom>
    </javadoc>
  </target>

  <target depends="jar,doc" name="zip">
    <echo message="${ant.project.name}: zip"/>
    <zip zipfile="${dirpackages}/${ant.project.name}.zip">
      <!-- bin/publish bin/client -->
      <zipfileset dir="." prefix="${ant.project.name}">
        <include name="${dirbin}/**/*.cmd"/>
      </zipfileset>
      <!-- doc/* -->
      <zipfileset dir="." prefix="${ant.project.name}">
        <include name="${dirdoc}/**/*"/>
      </zipfileset>
      <!-- etc/* -->
      <zipfileset dir="." prefix="${ant.project.name}">
        <include name="${diretc}/**/*"/>
      </zipfileset>
      <!-- lib/GenericMigrate-0-0.jar, lib/JAXWS JARs incl. endorsed -->
      <zipfileset dir="${dirbuild}" prefix="${ant.project.name}">
        <include name="**/*.jar"/>
      </zipfileset>
      <!-- readme.txt etc. -->
      <zipfileset dir="." prefix="${ant.project.name}">
        <include name="*.txt"/>
      </zipfileset>
    </zip>
  </target>

  <target depends="jar" name="war">
    <echo message="${ant.project.name}: war"/>
    <war warfile="${dirpackages}/${ant.project.name}.war"
        manifest="${filewebmanifest}"
        webxml="${dirwebinf}/${filewebxml}">
      <lib dir="${dirbuildlib}" includes="${ant.project.name}.jar"/>
      <webinf dir="${dirwebinf}">
        <include name="${filesunjaxws}"/>
      </webinf>
    </war>
  </target>

  <target depends="war,zip" name="packages"/>

</project>
```

The resulting packages *GenericMigrate-0-0.zip* (for stand-lone usage) and *GenericMigrate-0.0.war* (for deployment in an application server) are being created in the *packages* folder of the project.

6.7 Deployment

The stand-alone package *GenericMigrate-0-0.zip* is deployed, by unpacking it anywhere. Double-clicking on *bin\publish.cmd* starts the service. Executing *bin\client.cmd* runs the client. The stand-alone package needs a *JAVA* run-time (*JAVA SE 1.5* or higher) installed and does not pollute the system with its endorsed folders. The client of the stand-alone package can also be used for accessing a *GenericMigrate-0-0* service entry anywhere, whether published locally or on the web, whether published stand-alone or in a container. If the service needs to be accessed through a proxy host, the *JAVA* networking can be directed to use such a proxy host by setting the environment variable *%JAVA_OPTS%*. E.g.

```
set JAVA_OPTS=-DproxyHost=www-proxy.admin.ch -DproxyPort=8080
```

6.8 What has been achieved so far

A packaging strategy has been decided upon. Following the strategy of IF today, we aim for loose coupling of PA/4 Web Services with the Integration Framework. We also aim for maximum usability of our service in all kinds of contexts without forcing our users to utilize a specific application server or a specific version of such a product.

We have developed a binary distribution of our Web Service that can be run anywhere, with just *JAVA* (1.5 or higher) installed. This binary distribution also contains a Web Client which can make use of the Web Service independent of its deployment. Thus the Web Client of the stand-alone distribution is equally able to address the Web Service deployed in a container like *Tomcat* or *JBoss*.

7 VERSIONING

Developers – as other people – sometimes become cleverer with time. Also the external requirements for a Web Service may change. New versions of a Tool for Objects may need to be supported. It is unavoidabla that at some point new versions of PA/4 services need to be made available. A new version of the *Migrate* interface within the IF has even more impact. In the PLANETS project we have experienced this kind of change of the IF interface, which made all efforts of the PA/4 developers obsolete.

We want to avoid such a situation in the future. Therefore we have to adopt a versioning strategy.

7.1 Goal and Strategy of Versioning

The goal of versioning is to avoid close coupling of service and clients. Thus a client or a service can be upgraded to a new version without breaking running applications.

The strategy, to achieve this goal, is that newly developed services and clients support both the current as well as one or more older versions.

We observe that the basic *WSDL* definition does not cater for versions of *WSDL* documents. Thus from the service point of view, new versions must be published as new *WSDL* documents with different names from the old version. The task of supporting both older and newer versions is achieved by publishing both service versions as separate services side by side.

The task of a client (e.g. some IF work flow) of these services is then to flexibly check whether a current version is available and using the fallback to an older version if that is not the case.

7.2 Version Development Guidelines

For this approach to work, it is necessary that the developers in the PLANETS project agree on some versioning guidelines.

We propose here, that a version consists of a major version number and a minor release number. We increase the release number whenever the service is upgraded (i.e. when its semantics change). We increase the version number when the interface (i.e. the *WSDL*, the syntax) is changed.

We append the version number to the *JAVA* interface (*SEI*) and the corresponding „Port“ name. We append both version and release numbers to the implementation (*SIB*) and the „Service“ name.

Web Service clients, by convention, always support both the current (at the time of development) version as well as one or more older versions, that are still supported. Web Services support the older versions by being published side by side with their older versions, which continue running until all clients have been upgraded.

The most basic rule that must be observed by all developers is that the major version number must be increased whenever the *WSDL* or its related *XSDs* (i.e. when the *JAVA* interface or the data type classes used in it) change. It is absolutely inacceptable to break old code by changing a *WSDL* document once it has been published and has clients programmed against it.

7.3 A Practical Example

We wish to approximate the *Migrate* interface of the IF rather than just migrating strings as parameters. Instead we will use a *DigitalObject* class as the type for the parameters of the input and output. This will change the *WSDL*. Therefore we start a new project with the name *GenericMigrate-1-0* with the interface *Migrate_1*.

```
/*== Migrate_1.java ================================================
PLANETS migration service.
Version     : $Id: Migrate_1.java 119 2009-07-17 07:48:53Z hartwig $
Application : PLANETS PA/4 migration services
Description : Migrate_1 defines a minimum interface for migration
Platform    : JAVA SE 1.5 or higher, JAX-WS2.1.5 (2008/10/30)
------------------------------------------------------------------------
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 07, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
==================================================================*/
package eu.planets_project.services.migrate;

import java.net.URI;
import java.util.List;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebParam;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

import eu.planets_project.services.datatypes.DigitalObject_1;
import eu.planets_project.services.datatypes.Parameter_1;

/*==================================================================*/
/** This interface corresponds to the service contract which is
 * represented by WSDL which is automatically generated by the annotation
 * processing of JAX-WS 2.1.5.
 * @author Hartwig Thomas <hartwig.thomas@enterag.ch>
 */
@WebService(targetNamespace=Migrate_1.sNS,
            name=Migrate_1.sNAME + "-" + Migrate_1.sVERSION)
public interface Migrate_1
{
  int iSTREAM_BUFFER_SIZE = 16240;
  String sNS = "http://migrate.services.planets-project.eu/";
  String sNAME = "Migrate";
  String sVERSION = "1";

  /*------------------------------------------------------------------*/
  /** transform the original (String) object and return the migrated object.
   * @param doRequest object to be migrated.
   * @param uriFormatOutput output format.
   * @param listParameters list of parameters.
   * @return the migrated (String) object.
   */
  @WebMethod(operationName="migrate")
  @RequestWrapper(localName="MigrateRequest-" + Migrate_1.sVERSION,
    targetNamespace=Migrate_1.sNS,
    className="MigrateRequest_"+Migrate_1.sVERSION)
  @ResponseWrapper(localName="MigrateResponse-" + Migrate_1.sVERSION,
    targetNamespace=Migrate_1.sNS,
    className="MigrateResponse_" + Migrate_1.sVERSION)
  @WebResult(name="mrResult") MigrateResult_1 migrate(
    @WebParam(name="doRequest") DigitalObject_1 doRequest,
    @WebParam(name="uriFormatOutput") URI uriFormatOutput,
    @WebParam(name="listParameters") List<Parameter_1>listParameters);

} /* interface Migrate_1 */
```

This interface approximates the current IF interface. We have cleaned up the referenced datatypes, which all are part of the interface, removing dead code stemming from automatic generation based on the abstract data model. An abstract data model should not be used for generating concrete code. Unexplained, unexplainable and unused data members are generally a burden on the code. Interfaces should be kept as small as possible. Non-interface tasks should be implemented in separate helper classes that are not part of the interface (e.g. even in out light version the implementations of the various static *getInstance* methods should be moved to separate helper classes). If it later becomes clear, that more members are needed, a new version of the interface can be introduced. That is precisely the idea behind a consequent application of versioning to the collective development of a large cooperating set of services and clients.

The following changes to the current IF interface were deliberate:

– The migrate() method does not need an input format parameter as that is already contained in the input *DigitalObject*.

– The *Parameter* type does not need a description, because it is never communicated from the service to the client.

– The *DigitalObject* type does not need a list of contained *DigitalObjects*, because it was decided in the workshop, that complex objects would be transmitted as ZIP files. Also the parallel MTOM-streaming of numerous contained objects is not recommended. Instead the zipped complex object is to be streamed as a single content.

It is possible, that our simplification went too far in some respects. We do not know about all uses of these types in different sub projects. However, the interfaces must be agreed upon by both parties (PA and IF) and they must avoid unused code.

It may be noted, that all datatypes used in the parameters of the service call also carry the version number. They are implicitly part of the interface. Thus any change in any of the dependent datatypes must be accompanied by an increase of the version number of the whole interface according to the versioning philosophy introduced here.

Is is also noteworthy that all names which influence the XML generation (*@XmlType* and *@XmlElement* annotations) are given explicitly. This ensures that different Web Service containers generate the same *WSDL* and *XSD* interface. On the other hand we keep the annotations at a minimum. Generally it is not a good idea to program via annotations, because their effects are usually not documented very unequivocally. (There is no effect of an annotation that could not have been achieved through plain programming.)

From a project management point of view, the migration service interfaces should have been agreed upon by both the IF and the PA sub projects and not just been generated from a dubious abstract data model. They also should not contain any platform dependence whatsoever. Finally, considering that they have to be marshalled by *JAXB* into an *XML* structure, which should be decodable by any Web Service client on any operating system platform and any programming environment, it is preferable, if the data types that are part of the Web Service interface be simple *JAVA* beans which translate into *XML* in a straight forward manner, thus being easily decodable by a *.NET* client, for example.

We also create an implementation of the generic Web Service which deserves, being called a generic. Every simple command-line tool can be implemented by this code by just exchanging the command-line actually executed.

Multiple Service Versions

Now we want to support both, the previous version of the service *GenericMigrate-0-0* as well as the new version *GenericMigrate-1-0*.

The client achieves the support of the older version by including its *JAR* file in the class path and referencing the old interface like this:

```
/*-------------------------------------------------------------------*/
/** migrate using a Migrate_1 service with fallback to Migrate_0
 * @param fileInput input file.
 * @param fileOutput output file.
 * @param sWsdlUrl URL of WSDL.
 * @param sService service name.
 * @return output text.
 */
private ServiceReport_1 migrate(
  File fileInput,
  File fileOutput,
  String sWsdlUrl,
  String sService)
  throws MalformedURLException, FileNotFoundException, IOException
{
  ServiceReport_1 sr = null;
  /* determine version of service */
  int iEnd = sService.lastIndexOf("-");
  int iStart = sService.substring(0, iEnd).lastIndexOf("-")+1;
  String sInterfaceVersion = sService.substring(iStart,iEnd);
  if (sInterfaceVersion.equals("1"))
  {
    Migrate_1 mig = getMigrate_1Proxy(sWsdlUrl, sService.replace('-', '_'));
    /* connect input file to data handler */
    DataHandler dhRequest = new DataHandler(new FileDataSource(fileInput));
    Content_1 content = Content_1.getInstance(dhRequest, null);
    DigitalObject_1 doRequest = DigitalObject_1.getInstance(
      null, null, content, null);
    /* run service */
    MigrateResult_1 mr = mig.migrate(doRequest, null, null);
    /* write output file from data handler */
    moveTo(mr.getDigitalObject().getContent().getValue(),fileOutput);
    sr = mr.getServiceReport();
  }
  else if (sInterfaceVersion.equals("0"))
  {
    Migrate_0 mig = getMigrate_0Proxy(sWsdlUrl, sService.replace('-', '_'));
```

```
      /* turn input file into a string */
      Reader rdr = new InputStreamReader(new FileInputStream(fileInput));
      StringBuilder sbInput = new StringBuilder();
      for (int iRead = rdr.read(); iRead != -1; iRead = rdr.read())
        sbInput.append((char)iRead);
      rdr.close();
      /* run service */
      String sOutput = mig.migrate(sbInput.toString());
      /* write resulting string to output file */
      Writer wtr = new OutputStreamWriter(new FileOutputStream(fileOutput));
      wtr.write(sOutput);
      wtr.close();
      /* generate empty service report */
      sr = ServiceReport_1.getInstance("Migrate_0 interface used.");
    }
    else
       throw new IllegalArgumentException(
         "Service interface must be Migrate_0 or Migrate_1!");
    return sr;
  } /* migrate */
```

Whenever the interface version is changed, the IF needs to do something like this, in order to prevent breaking all services implementing older interfaces.

if we publish the service on an application server, supporting both versions is trivial. One just deploys both *WAR* files as Web Services side by side. Thus clients relying on the old version will still find it in place.

Our own light-weight publisher, based on Sun‘s *HttpServer* class unfortunately cannot bind more than one endpoint to the server. This shows, that it cannot really be used for production strength deployment.

8 LARGE OBJECTS

Before we actually publish the *Migrate_1* interface, we should prefer a version that supports large files. The versions displayed up to now load the whole content of the digital object into memory, before applying the transformation. Instead, we wish to pass the byte stream represented as a *DataHandler* as an attachment of the *SOAP* message and stream it to a temporary file as we receive it through the socket‘s input stream. Similarly we want to stream the migrated file to the socket‘s output stream, without loading it into memory. Of course, the client application will have to implement the same operations.

In order to support large objects we introduce the *MTOM* (Message Transmission Optimization Mechanism) annotation, indicating that we wish to turn large parameters into SOAP attachments. In addition we will have to use the *StreamingDataHandler* to effect the streaming. Finally we have to implement a somewhat special *TempFileDataSource* which will delete the data source‘s file, once it is closed. This prevents temporary files from cluttering up the server.

8.1 MTOM and Streaming

As we have not published the interface and nobody else depends on it yet, we may keep the version number for this change.

The final *Migrate_1* interface remains unchanged. The *SIB GenericMigrate_1_0* is embellished with an *MTOM* annotation. Its *DataHandler* is treated as a *StreamingDataHandler*.

```
/*== GenericMigrate_1_0.java ==========================================
Generic PLANETS migration service implementation.
Version     : $Id: GenericMigrate_1_0.java 137 2009-07-17 17:35:45Z hartwig $
Application : PLANETS PA/4 migration services
Description : GenericMigrate_1_0 implements Migrate_1 and executes
              a command-line tool turning the request object
              into a result object.
Platform    : JAVA SE 1.5 or higher, JAX-WS2.1.5 (2008/10/30)
-----------------------------------------------------------------------
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 07, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
=====================================================================*/
package eu.planets_project.services.migration.generic;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.text.MessageFormat;
import java.util.List;
import java.util.logging.Logger;

import javax.activation.DataHandler;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;
import com.sun.xml.ws.developer.StreamingDataHandler;

import eu.planets_project.services.datatypes.Content_1;
import eu.planets_project.services.datatypes.DigitalObject_1;
import eu.planets_project.services.datatypes.Parameter_1;
import eu.planets_project.services.datatypes.ServiceReport_1;
import eu.planets_project.services.utils.TempFileDataSource;
import eu.planets_project.services.migrate.Migrate_1;
import eu.planets_project.services.migrate.MigrateResult_1;
import eu.planets_project.services.utils.CommandExec;

/*====================================================================*/
/** This class is the bean (SIB) that implements the Migrate
 * interface (SEI).
 * @author Hartwig Thomas <hartwig.thomas@enterag.ch>
 */
@WebService(
  endpointInterface = "eu.planets_project.services.migrate.Migrate_1",
  portName=Migrate_1.sNAME + "-" + Migrate_1.sVERSION,
  targetNamespace=Migrate_1.sNS,
  serviceName=GenericMigrate_1_0.sNAME + "-" + GenericMigrate_1_0.sVERSION)
@MTOM(enabled=true, threshold=Migrate_1.iSTREAM_BUFFER_SIZE)
public class GenericMigrate_1_0
  implements Migrate_1
{
  public static final String sNS =
    "http://generic.migration.services.planets-project.eu/";
  public static final String sNAME = "GenericMigrate";
  public static final String sVERSION = Migrate_1.sVERSION + "-0";

  private Logger m_log = Logger.getLogger(GenericMigrate_1_0.class.getName());
  private static final String sCOMMAND_TEMPLATE = "cmd /C copy /b \"{0}\" \"{1}\"";
  protected String getCommandTemplate() { return sCOMMAND_TEMPLATE; }
  private static final long lMS_TIMEOUT = 3600000; /* 1 hour in ms */
  protected long getMsTimeout() { return lMS_TIMEOUT; }
  private static final String sINPUT_EXTENSION = ".dat";
  protected String getInputExtension() { return sINPUT_EXTENSION; }
  private static final String sOUTPUT_EXTENSION = ".dat";
  protected String getOutputExtension() { return sOUTPUT_EXTENSION; }

  /*------------------------------------------------------------------*/
  /** creates a generic migrate command from resources, input and output file.
   * @param fileInput input file.
   * @param fileOutput output file.
   * @return command line to be executed.
   */
  protected String createCommand(File fileInput, File fileOutput)
    throws IOException
  {
    String sCommand = MessageFormat.format(sCOMMAND_TEMPLATE, new Object[] {
      fileInput.getAbsolutePath(),
      fileOutput.getAbsolutePath()});
    return sCommand;
  } /* createCommand */

  /*------------------------------------------------------------------*/
  /** executes the given command, possibly terminating with a timeout.
   * @param sCommand command.
   * @param lMsTimeout timeout in milliseconds.
   * @return service report with exit code, stdout and stderr.
   * @throws IOException if an I/O error occurred,
   *         InterruptedException if process was interrupted
   *         or a timeout occurred.
   */
  private ServiceReport_1 executeCommand(String sCommand, long lMsTimeout)
    throws IOException, InterruptedException
  {
    ServiceReport_1 sr = null;
    CommandExec ce = new CommandExec();
    try
    {
      int iResult = ce.execute(sCommand, lMsTimeout);
      /* we make the assumption, that external commands return 0 on success */
      if (iResult == ServiceReport_1.STATE_SUCCESS)
        sr = ServiceReport_1.getInstance(ce.getOutput(), ce.getError());
      else
        sr = ServiceReport_1.getInstance(ce.getOutput(), ce.getError(), iResult);
    }
    catch(IOException ie)
    { m_log.severe(ie.getClass().getName()+": "+ie.getMessage()); }
    catch(InterruptedException ie)
    { m_log.severe(ie.getClass().getName()+": "+ie.getMessage()); }
    return sr;
  } /* executeCommand */

  /*------------------------------------------------------------------*/
  /* (non-Javadoc)
   * @see eu.planets_project.services.migrate.Migrate_1#migrate(java.lang.String)
   * migrate implements a command-line from the resources.
   */
  public MigrateResult_1 migrate(
    DigitalObject_1 doRequest,
    URI uriFormatOutput,
    List<Parameter_1>listParameters)
  {
    m_log.fine("Classpath: "+System.getProperty("java.class.path"));
    ServiceReport_1 sr = null;
    DigitalObject_1 doResponse = null;
    try
    {
      /* save content of doRequest as a temporary input file */
      File fileInput = File.createTempFile("mig", getInputExtension());
      /* make sure, input at least gets removed when the service is stopped */
      fileInput.deleteOnExit();
      /* redirect streaming data handler to this file */
      DataHandler dhInput = doRequest.getContent().getValue();
      if (dhInput instanceof StreamingDataHandler)
      {
        m_log.info("Streaming to "+fileInput.getAbsolutePath());
        StreamingDataHandler sdhInput = (StreamingDataHandler)dhInput;
        sdhInput.moveTo(fileInput);
        sdhInput.close();
      }
      else // appears to be superfluous
      {
        m_log.info("Copying to "+fileInput.getAbsolutePath());
        FileOutputStream fos = new FileOutputStream(fileInput);
        InputStream is = dhInput.getInputStream();
        byte[] buf = new byte[Migrate_1.iSTREAM_BUFFER_SIZE];
        for (int iRead = is.read(buf); iRead != -1; iRead = is.read(buf))
          fos.write(buf,0,iRead);
        is.close();
        fos.close();
      }
      m_log.info("Received "+String.valueOf(fileInput.length())+" bytes.");
      /* we ignore all parameters for now: Timeout might become a parameter later */
      File fileOutput = File.createTempFile("mig", getOutputExtension());
      /* create the command-line from the file names and the parameters */
      String sCommand = createCommand(fileInput, fileOutput);
      m_log.info("Command: "+sCommand);
      sr = executeCommand(sCommand,getMsTimeout());
      m_log.info("Migrated content to : "+fileOutput.getAbsolutePath());
      fileInput.delete();
      m_log.info("File "+fileInput.getAbsolutePath()+" deleted.");
      /* make sure, output at least gets removed when the service is stopped */
      fileOutput.deleteOnExit();
      m_log.info("Sending "+String.valueOf(fileOutput.length())+" bytes.");
      /* load content of doResponse from the temporary output file */
      DataHandler dhResponse = new DataHandler(new TempFileDataSource(fileOutput));
      Content_1 content = Content_1.getInstance(dhResponse, null);
      doResponse = DigitalObject_1.getInstance(null, uriFormatOutput, content, null);
    }
    catch(Exception e)
    {
      sr = ServiceReport_1.getInstance(
        e.getClass().getName()+": "+e.getMessage(),
        ServiceReport_1.STATE_ERROR);
    }
    m_log.info("returning result");
    return MigrateResult_1.getInstance(doResponse, sr);
  } /* migrate */

} /* GenericMigrate_1_0 */
```

The *MTOM* is enabled by the single *@MTOM* annotation of the class, which moves large objects to an attachment of the *SOAP* message, if they are larger than the threshold given (here: 16240 bytes). Streaming without storing all data in memory is implemented by making use of the *Streaming-DataHandler*'s *moveTo* method.

The whole issue of streaming large files as attachments is rather ticklish. Any other annotation (*@BindigType, @StreamingAttachment, StreamingAttachmentFeature, MTOMFeature,* ...) tends to break the streaming of *JAX-WS 2.1.5*. It is therefore strongly recommended to stick to the annotations presented here.

8.2 A Read-Once DataSource

The read-once *FileDataSource* is implemented in the class *TempFileDataSource*. This obviates the need for complex handling of „asynchronous" requests.

```
/*== TempFileDataSource.java ==========================================
FileDataSource for "read-once" files that are to be deleted after usage.
Version     : $Id: TempFileDataSource.java 134 2009-07-17 17:11:26Z hartwig $
Application : PLANETS PA/4 migration services
Description : TempFileDataSource implements a FileDataSource
              for streaming which deletes the file when it is
              closed.
Platform    : JAVA SE 1.5 or higher, JAX-WS2.1.5 (2008/10/30)
-----------------------------------------------------------------------
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 16, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
=====================================================================*/
package eu.planets_project.services.utils;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.util.logging.Logger;

import javax.activation.FileDataSource;

/*===================================================================*/
/** This class implements a FileDataSource which deletes the file
 * when it is closed.
 * @author Hartwig Thomas
 */
public class TempFileDataSource
  extends FileDataSource
{
  private final static String sSTREAM_MIME_TYPE = "application/octet-stream";
  private Logger m_log = Logger.getLogger(TempFileDataSource.class.getName());
  private final File m_fileSource;

  /*-----------------------------------------------------------------*/
  /** constructor
   * @param fileSource input file which will be deleted on close.
   */
  public TempFileDataSource(File fileSource)
  {
    super(fileSource);
    m_fileSource = fileSource;
  } /* constructor TempFileDataSource */

  /*-----------------------------------------------------------------*/
  /** open the file as an input stream
   * @return open input stream which deletes the file on close.
   * @throws FileNotFoundException if the file cannot be opened.
   */
  @Override
  public InputStream getInputStream()
```

```
    throws FileNotFoundException
  {
    m_log.info("getInputStream");
    return new TempFileInputStream(m_fileSource);
  } /* getInputStream */

  /*------------------------------------------------------------------*/
  /** MIME type of file stream is used for streaming support.
   * @return application/octet-stream.
   */
  @Override
  public String getContentType()
  {
    m_log.info("Content-type: "+sSTREAM_MIME_TYPE);
    return sSTREAM_MIME_TYPE;
  } /* getContentType */

  /*------------------------------------------------------------------*/
  /** file name is returned as name of DataSource.
   * (Possibly super.getName() returns the same result.)
   * @return file name.
   */
  @Override
  public String getName()
  {
    m_log.info("Name: "+m_fileSource.getPath());
    return m_fileSource.getPath();
  } /* getName */

  /*------------------------------------------------------------------*/
  /** no output stream.
   * (Probably super.getOutputStream() returns the same result.)
   * @return null.
   */
  @Override
  public OutputStream getOutputStream()
  {
    return null;
  } /* getOutputStream */

  /*==================================================================*/
  /** inner class that handles the delete on close.
   * Most of the code is just here to log the activity on this InputStream.
   */
  private static final class TempFileInputStream
    extends FileInputStream
  {
    private Logger m_log = Logger.getLogger(TempFileInputStream.class.getName());
    /** the file to be read and deleted on close */
    private File m_file;
    /** the number of bytes read */
    private long m_lRead;

    /*------------------------------------------------------------------*/
    /** constructor
     * @param file the file to be opened and deleted on close.
     * @return open input stream
     * @throws FileNotFoundException if file cannot be opened.
     */
    public TempFileInputStream(File file)
      throws FileNotFoundException
    {
      super(file);
      m_file = file;
      m_lRead = 0;
    } /* constructor TempFileInputStream */

    /*------------------------------------------------------------------*/
    /** closes the input stream and deletes the file.
     * @throws IOException if an I/O exception occurred.
     */
    @Override
    public void close()
      throws IOException
    {
      super.close();
      m_log.info("close");
      if (m_file != null)
      {
        if (m_lRead != m_file.length())
          m_log.info("File of size " + String.valueOf(m_file.length()) +
              " was closed after reading " + String.valueOf(m_lRead) + " bytes.");
        m_file.delete();
        if (!m_file.exists())
          m_log.info("File "+m_file.getAbsolutePath()+" deleted.");
        m_file = null;
      }
    } /* close */

    /*--------------------------------------------------------------------*/
    /** reads a byte and keeps count of number of bytes read.
     * @throws IOException if an I/O exception occurred.
     */
    @Override
    public int read()
      throws IOException
    {
      int iRead = super.read();
      if (iRead != -1)
        m_lRead++;
      else
        m_log.info("EOF in read() reached.");
      return iRead;
    } /* read */

    /*--------------------------------------------------------------------*/
    /** reads a buffer and keeps count of number of bytes read.
     * @throws IOException if an I/O exception occurred.
     */
    @Override
    public int read(byte[] buffer)
      throws IOException
    {
      int iRead = super.read(buffer);
      if (iRead != -1)
        m_lRead += iRead;
      else
        m_log.info("EOF in read(buffer) reached.");
      return iRead;
    } /* read */

    /*--------------------------------------------------------------------*/
    /** reads a partial buffer and keeps count of number of bytes read.
     * @throws IOException if an I/O exception occurred.
     */
    @Override
    public int read(byte[] buffer, int iOffset, int iLength)
      throws IOException
    {
      int iRead = super.read(buffer,iOffset,iLength);
      if (iRead != -1)
        m_lRead += iRead;
      else
        m_log.info("EOF in read(buffer,offset,length) reached.");
      return iRead;
    } /* read */

  } /* class TempFileInputStream */

} /* TempFileDataSource */
```

This class just deletes the input file, once it has been read and closed.

Utility CommandExec

It may be noted, that we make use of the utility class *CommandExec* rather than the existing IF tools based of *ProcessBuilder*. We have two reasons for avoiding the utility class: on the one hand, we prefer simple classes that do few things. On the other hand, we foresee, that we may need to change it in the future, when we wrap other tools. We may have to implement a timeout for tools that may under special circumstances not terminate correctly. We certainly will have to implement a limitation on the size of the buffers for *stdout* and *stderr*, that may become too large for being contained in *JAVA* memory (e.g. keeping the the first and the last few lines).

Especially for wrapping externally executing commands this class is very central for PA/4.

8.3 A Streaming Client

Of course, the client code needs to implement the same data handling in the opposite order. In it we also must enable *MTOM* and we must tell the underlying *HTTP* handler to enable „chunking“.

```
...

  /*--------------------------------------------------------------------*/
  /** getMigrate_1Proxy returns a proxy class implementing the Migrate_1
   * interface and representing the Web Service.
   * @param sWsdlUrl URL of WSDL.
   * @param sService service name.
   * @return output text.
   */
  private Migrate_1 getMigrate_1Proxy(String sWsdlUrl, String sService)
    throws MalformedURLException
  {
    URL urlWsdl = new URL(sWsdlUrl);
    Migrate_1 migProxy = null;
    /* service */
    Service service = Service.create(urlWsdl, new QName(Migrate_1.sNS,sService));
    /* port corresponds to the interface */
    migProxy = (Migrate_1)service.getPort(Migrate_1.class);
    System.out.printf("Created proxy class for service %s\n",service.getServiceName());
    BindingProvider bp = (BindingProvider)migProxy;
    SOAPBinding sb = (SOAPBinding)bp.getBinding();
    /* enable MTOM for the interface */
    sb.setMTOMEnabled(true);
    if (sb.isMTOMEnabled())
      System.out.printf("MTOM enabled\n\n");
    /* enable streaming for the interface */
    Map<String,Object> mapContext = bp.getRequestContext();
                 mapContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE,    new
Integer(Migrate_1.iSTREAM_BUFFER_SIZE));
    return migProxy;
  } /* getMigrate_1Proxy */

...

  /*--------------------------------------------------------------------*/
  /** migrate using a Migrate_1 service with fallback to Migrate_0
   * @param fileInput input file.
   * @param fileOutput output file.
   * @param sWsdlUrl URL of WSDL.
   * @param sService service name.
   * @return output text.
   */
  private ServiceReport_1 migrate(
    File fileInput,
    File fileOutput,
    String sWsdlUrl,
    String sService)
    throws MalformedURLException, FileNotFoundException, IOException
  {
    ServiceReport_1 sr = null;
    /* determine version of service */
    int iEnd = sService.lastIndexOf("-");
    int iStart = sService.substring(0, iEnd).lastIndexOf("-")+1;
    String sInterfaceVersion = sService.substring(iStart,iEnd);
    if (sInterfaceVersion.equals("1"))
    {
      Migrate_1 mig = getMigrate_1Proxy(sWsdlUrl, sService);
      /* connect input file to data handler */
      System.out.println("Sending "+String.valueOf(fileInput.length())+" bytes.");
      DataHandler dhRequest = new DataHandler(new FileDataSource(fileInput));
      Content_1 content = Content_1.getInstance(dhRequest, null);
      DigitalObject_1 doRequest = DigitalObject_1.getInstance(
        null, null, content, null);
      /* run service */
      System.out.println("Calling service ...");
      MigrateResult_1 mr = null;
      try { mr = mig.migrate(doRequest, null, null); }
      catch(WebServiceException we)
      {
        /* try again without chunking */
        BindingProvider bp = (BindingProvider)mig;
        /* enable streaming for the interface */
        Map<String,Object> mapContext = bp.getRequestContext();
        mapContext.put(JAXWSProperties.HTTP_CLIENT_STREAMING_CHUNK_SIZE, null);
        try
        {
          System.out.println("... without support for streaming large files ...");
          mr = mig.migrate(doRequest, null, null);
        }
        catch(Exception e)
        {
          System.out.println(e.getClass().getName()+": "+e.getMessage() +
            " ("+e.getCause()+")");
          e.printStackTrace();
        }
      }
      if (mr != null)
      {
        System.out.println("... result received.");
        /* write output file from data handler */
        DataHandler dhOutput = mr.getDigitalObject().getContent().getValue();
        if (dhOutput instanceof StreamingDataHandler)
        {
          System.out.println("Streaming to "+fileOutput.getAbsolutePath());
          StreamingDataHandler sdhResponse =
           (StreamingDataHandler)mr.getDigitalObject().getContent().getValue();
          sdhResponse.moveTo(fileOutput);
          sdhResponse.close();
        }
        else
        {
          System.out.println("Copying to "+fileOutput.getAbsolutePath());
          FileOutputStream fos = new FileOutputStream(fileOutput);
          InputStream is = dhOutput.getInputStream();
          byte[] buf = new byte[Migrate_1.iSTREAM_BUFFER_SIZE];
          for (int iRead = is.read(buf); iRead != -1; iRead = is.read(buf))
            fos.write(buf,0,iRead);
          is.close();
          fos.close();
        }
        System.out.println("Received "+String.valueOf(fileOutput.length())+" bytes.");
        sr = mr.getServiceReport();
      }
    }
    else if (sInterfaceVersion.equals("0"))
    {
      ...
    }
    else
      throw new IllegalArgumentException(
        "Service interface must be Migrate_0 or Migrate_1!");
    return sr;
  } /* migrate */

...
```

Again the *MTOM*-handling is rather brittle. Even small deviations from it break the mechanism. As it stands it will enable streaming large files to the service installed in an application server. The stand-alone publisher unfortunately does not support chunking. When transmission to it is started, it throws an exception. Therefore the client code catches a *WebExecption* and tries the transmission again with chunking turned off. This way it will succeed as a client to a stand-alone instance of the service, albeit without streaming support and thus limited to small files.

8.4 What has been achieved so far

With these changes we can now use the Web Service *GenericMigrate-1-0* for migrating files of any size.

9 MDB2SIARDMIGRATE

The implementation of *Mdb2SiardMigrate-1-0* – as that of most other wrappers of command-line tools – becomes very simple.

One implements an *SIB* class which extends the *GenericMigrate_1_0* and only defines the correct *WebService* name:

```
/*== Mdb2SiardMigrate_1_0.java ========================================
PLANETS migration service implementation migrating MDB files to SIARD files.
Version     : $Id: Mdb2SiardMigrate_1_0.java 143 2009-07-20 08:41:48Z hartwig $
Application : PLANETS PA/4 migration services
Description : Mdb2SiardMigrate_1_0 extends GenericMigrate_1_0 and executes
              the command-line tool convmdb turning the request object
              into a result object.
Platform    : JAVA SE 1.5 or higher, JAX-WS2.1.5 (2008/10/30)
------------------------------------------------------------------------
Copyright  : Swiss Federal Archives, Berne, Switzerland
             (pending PLANETS copyright agreements)
Created    : July 07, 2009, Hartwig Thomas, Enter AG, Zurich
Sponsor    : Swiss Federal Archives, Berne, Switzerland
======================================================================*/
package eu.planets_project.services.migration.mdb2siard;

import java.io.File;
import java.io.IOException;
import java.text.MessageFormat;
import java.util.logging.Logger;

import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;

import eu.planets_project.services.migrate.Migrate_1;
import eu.planets_project.services.migration.generic.GenericMigrate_1_0;

/*====================================================================*/
/** this class extends GenericMigrate_1_0 and executes the command-line
 * tool convmdb turning the request object into a result object.
 * @author Hartwig Thomas <hartwig.thomas@enterag.ch>
 */
@WebService(
    endpointInterface = "eu.planets_project.services.migrate.Migrate_1",
    portName=Migrate_1.sNAME+"-"+Migrate_1.sVERSION,
    targetNamespace=Migrate_1.sNS,
    serviceName=Mdb2SiardMigrate_1_0.sNAME + "-" + Mdb2SiardMigrate_1_0.sVERSION)
  @MTOM(enabled=true, threshold=Migrate_1.iSTREAM_BUFFER_SIZE)
public class Mdb2SiardMigrate_1_0
  extends GenericMigrate_1_0 /* and therefore implements Migrate_1 */
{
  private Logger m_log = Logger.getLogger(Mdb2SiardMigrate_1_0.class.getName());
                 public        static        final        String        sNS        =
"http://mdb2siard.migration.services.planets_project.eu/";
  public static final String sNAME = "Mdb2SiardMigrate";
  public static final String sVERSION = Migrate_1.sVERSION + "-0";

  private static final String sCOMMAND_TEMPLATE =
    "convmdb.cmd /dsn:{0} \"{1}\" \"{2}\"";
  @Override
  protected String getCommandTemplate() { return sCOMMAND_TEMPLATE; }
  private static final long lMS_TIMEOUT = 3600000; /* 1 hour in ms */
  @Override
  protected long getMsTimeout() { return lMS_TIMEOUT; }
  private static final String sINPUT_EXTENSION = ".mdb";
  @Override
  protected String getInputExtension() { return sINPUT_EXTENSION; }
  private static final String sOUTPUT_EXTENSION = ".siard";
  @Override
  protected String getOutputExtension() { return sOUTPUT_EXTENSION; }

  /*------------------------------------------------------------------*/
  /** creates a generic migrate command from resources, input and output file.
   * @param fileInput input file.
   * @param fileOutput output file.
   * @return command line to be executed.
   */
  @Override
  protected String createCommand(File fileInput, File fileOutput)
    throws IOException
  {
    /* compute DSN from input file name */
    String sDsn = fileInput.getName();
    sDsn = sDsn.substring(0,sDsn.length()-getInputExtension().length());
    m_log.info("DSN: "+sDsn);
    String sCommand = MessageFormat.format(sCOMMAND_TEMPLATE,
      new Object[] {
        sDsn,
        fileInput.getAbsolutePath(),
        fileOutput.getAbsolutePath()}
      );
    return sCommand;
  } /* createCommand */

} /* class Mdb2SiardMigrate_1_0 */
```

This just sets up the external command to be executed. All the rest is done by *GenericMigrate_1_0*.

The publisher needs to mention the correct class to be published.

A major addition is a folder *tools* in the ZIP file, which contains the tool which is called on the command-line. This must be installed prior to publishing the service. As it is difficult to communicate configurable path names into the application servers, we stipulate, that the tools are properly listed in the *PATH* environment variable.

9.1 What has been achieved so far

By deriving from *GenericMigrate_1_0* we can wrap external command-line tools very easily. They can be deployed as *WAR* files in different application servers. They can handle large files in a synchronous fashion.

9.2 What needs to be done next

Coordination with IF

Before many new features are added on this basis, the PA subproject needs to be coordinated with the IF subproject with respect to

– Migrate interface: common understanding

– Versioning approach

Complex Objects

Complex objects (e.g. .MSG files with attachments) can probably be handled like simple objects, once the decision was made, that all complex objects are zipped. The wrapping of a .MSG file conversion would just produce a .ZIP file. The conversion of a ZIP file would be handled within the IF work flow by recursively launching all necessary conversions on each file contained in the ZIP file.

Referenced Objects

Referenced objects (e.g. Oracle databases) can be handled without any changes to the *Migrate* interface. One just transmits the URI and an empty byte stream. The only question to be decided is, whether *DbUser* and *Db-Password* should go into the request *URI* (query portion) or be passed as *Parameters*.

Zurich, July 20, 2009