JOTM-BTP: a BTP extension for JOTM

Jeff Mesnil

May 15, 2003

Abstract

This guide describes the BTP extension of JOTM.

Contents

1 Introduction

This guide describes JOTM-BTP, the BTP extension of JOTM. It was contributed by Pierre-Yves Gibello and Xavier Spengler of ExperLog.
BTP (Business transaction Protocol) is a standard defined by the OASIS Consortium. BTP goal is to define XML-based technology for business transactions on the Internet. Business to business interactions on the Internet pose unique challenges; including transactions that span multiple enterprises and long lasting transactions. The interdependent workflows among multiple trading partners, which drive business transactions, need to be coordinated to ensure that the outcome of the transaction is reliable.
If you have any questions or comments on this BTP extension, do no hesitate to let us (`mailto:jotm@objectweb.org`) know.

2 Prerequisites

In order to install or use this extension, you will need a few Java application which you may already have:

- Ant 1.5.x
- Tomcat 4.1.x
- Axis 1.0
- JOTM 1.4

2.1 Ant

JOTM-BTP uses Ant (version 1.5) for its build process.
Ant can be downloaded from `http://jakarta.apache.org/ant/`. See the documentation of Ant to set it up (`http://jakarta.apache.org/ant/manual/`).

2.2 Tomcat

Tomcat is the web container developed by Apache/Jakarta. JOTM-BTP uses Tomcat 4.1.x which can be downloaded from `http://jakarta.apache.org/tomcat/`. See the documentation of Tomcat 4.1 to set it up (`http://jakarta.apache.org/tomcat/tomcat-4.1-doc/`).
JOTM-BTP build process relies on the `CATALINA_HOME` environment property which has to be set to the directory where tomcat is installed:

```
$ export CATALINA_HOME=<path_to_tomcat_directory>
```

2.3 Axis

Axis is a project of Apache/XML which implements the SOAP (Simple Access Object Protocol). JOTM-BTP uses Axis 1.0 which can be downloaded from `http://xml.apache.org/axis/releases.html`. Once Axis package has been downloaded and unzipped, copy the `webapps/axis/` directory into the `$CATALINA_HOME/webapps/` directory. JOTM-BTP build process expects to find axis webapp in the `$CATALINA_HOME/webapps/` directory. If it doesn't, build process will fail.

2.4 JOTM

To download and install JOTM, please refer to JOTM installation guide.

3 Installing JOTM-BTP

JOTM-BTP is available under two configurations:

- a package including jars and examples: `jotm-btp-x.y.tgz` where `x.y` is the version of JOTM-BTP
- from CVS

3.1 From a package

JOTM-BTP package can be downloaded from JOTM SourceForge page (`http://debian-sf.objectweb.org/projects/jotm/`).
To install JOTM-BTP from a package, unzip the file with **gunzip** and **tar** on Unix systems and **winzip** on Windows.
This will create a new directory `jotm-btp-x.y/`.

3.2 From CVS

CVS provides network-transaparent source control for groups of developers (more info at `http://www.cyclic.com`).
Working with CVS allows you to do things like **cvs status** or **cvs diff** and any other read-only CVS command.
To get JOTM from CVS, type:

```
$ cvs -d :pserver:anonymous@cvs.objectweb.org:/cvs/JOTM login
$ CVS password: [type Enter]
$ cvs -d :pserver:anonymous@cvs.objectweb.org:/cvs/JOTM co -P jotm-btp
```

This will create a working repository of JOTM in the `jotm-btp/` directory.

4 Ant commands

If you've retrieved JOTM-BTP from CVS, you'll have to build it and create a distribution before using it.
JOTM-BTP relies on Ant for its build process.
All Ant commands are to be typed in the `jotm-btp/` directory (i.e. in the same directory than the `build.xml` file).
To have a list and descriptions of all Ant available targets for JOTM, type:

```
$ ant -projecthelp
```

Before executing any Ant commands, be sure that:

- `CATALINA_HOME` has been set
- Axiz webapp has been placed in the `webapps/` directory of Tomcat
- The relative path to JOTM distribution is correct in `jotm-btp/build.properties` (by default, the path is correct if you're using both JOTM and JOTM-BTP form CVS. Otherwise, you may have to change it)

Ant messages will warn you if one of these 3 conditions is not met.

4.1 Compile and build JOTM-BTP

This step is necessary only if you have modified JOTM-BTP source code or you've retrieved source files from CVS.
You just have to type:

```
$ ant dist
```

This will compile all JOTM-BTP source files and create all the jar files.
A working version of JOTM-BTP is now available in the `output/dist/` subdirectory (the so-called *distribution* directory of JOTM-BTP).

4.2 Generate Javadoc

To generate JOTM-BTP Javadoc, simply type:

```
$ ant jdoc
```

Javadoc pages are now browsable from the `output/dist/jdoc/` subdirectory.

4.3 Generate Documentation

JOTM-BTP documentation is written in LaTeX.
We use **pdflatex** tool to generate PDF files and **latex2html** to generate HTML files. Since these two tools may not be installed on your system, it's up to you to inform Ant that it'll have the tools to perform document generation.
PDF (resp. HTML) generation is triggered by `pdlatex` (resp. `latex2html`) property on the command line. What's more, for HTML generation a shell script, **doc2html**, is used. So for the moment, you can generate HTML documentation only from Linux. Sorry... (Anyway, documentation is still available online at `http://www.objectweb.org/jotm/doc/` in both PDF and HTML format).

- If you've **pdflatex** on your system, you can ask Ant to generate PDF documentation by typing

  ```
  $ ant doc -Dpdflatex=1
  ```

- If you've **latex2html** and you're on *Linux*, you can ask Ant to generate HTML documentation by typing

  ```
  $ and doc -Dlatex2html=1
  ```

Of course, you can also do both

```
$ and doc -Dpdflatex=1 -Dlatex2html=1
```

Generated document is put in the `output/dist/doc/` directory.

4.4 Clean JOTM-BTP

To remove files generated during compilation or build process, type:

```
$ ant clean
```

You'll start from a clean working directory again.

4.5 Source Structure

When you retrieve JOTM-BTP source, the structure of your `jotm-btp/` directory is the following one:

- `GettingStarted.txt` - a reminder of the main Ant targets for JOTM-BTP
- `README.txt` - the usual README file
- `build.xml` - the main Ant build file used by JOTM-BTP
- `build.properties` - configuration of Ant properties are done in this file
- `doc/` - all documentation sources are in this directory (e.g. `install.tex` from which this guide has been generated)
- `externals/` - some external libraries used by JOTM (which are not part of JOTM, Tomcat or Axis)
- `src/` - source code of JOTM-BTP
- `demo/` - JOTM-BTP demonstration (based on booking of hotel and flight) source code

4.6 Distribution Structure

Once you've built JOTM-BTP (i.e. in the `output/dist/` subdirectory) or if you retrieved it from its package, the ditribution structure of JOTM-BTP is the following:

- `README.txt` - still the usual README file
- `doc/` - this directory may contain PDF and HTML version of the various documentation files (see section 4.3)
- `demo/` - JOTM-BTP demonstration (binary files, Javadoc)
- `jdoc/` - Javadoc of JOTM-BTP
- `lib/` - JOTM-BTP library directory (it contains JOTM-BTP jar file and some other used jar files)

5 BTP Demonstration

Once you have a *distribution* of JOTM-BTP, one handy way to be sure that it is working is to run the demonstration (which is in the `demo/` directory of the distribution).

5.1 Scenario

The BTP demonstration is based on a common transaction example: a travel agency which books both one hotel and one flight in the same transaction. The demonstration is composed of 3 Web services:

- `TravelAgency`
- `HotelReservation`
- `FlightReservation`

The `TravelAgency` will contact both `HotelReservation` and `FlightReservation` to book them in the same transaction. If a problem occurs (such as no more place in the hotel or in the flight), the transaction will be rolled back for both. Otherwise, a reservation will be kept for both of them under the same transaction ID.

In addition to the travel agency client, there are two other servlets which can be used to supervise SOAP messages, transaction messages, hotel and flight reservation,...

5.2 Installation

Before installing the demonstration, you'll need to have a correct installation of both Tomcat and Axis (see section 2).

5.2.1 JOTM and JOTM-BTP

First, copy into `$CATALINA_HOME/webapps/axis/WEB-INF/lib/` directory:

- Jar files from JOTM distribution (in `$JOTM_HOME/lib/` directory):
  - `jotm.jar`
  - `jotm_jrmp_stubs.jar`
  - `jonas_timer.jar`
  - `jta_spec1_0_1.jar` (this jar is needed due to internal dependencies in JOTM on JTA interfaces)
- Jar files from JOTM-BTP distribution:
  - `jotm-btp.jar`
  - `kxml2.jar`

5.3 Deployment

Still into `$CATALINA_HOME/webapps/axis/WEB-INF/lib/` directory, copy:

- `demo/btp-demo.jar`

Copy into `$CATALINA_HOME/webapps/axis/`:

- `demo/btp.html`

5.3.1 Web Services deployment

You can find in the `demo/` directory 2 `.wsdd` files (`deploy.wsdd` and `undeploy.wsdd`). `deploy.wsdd` (resp. `undeploy.wsdd`) is used to deploy (resp. undeploy) our Demonstration Web Service into Axis.

To deploy, in the `demo/` directory, type:

```
$ java org.apache.axis.client.AdminClient deploy.wsdd
```

See Axis documentation to set the CLASSPATH before deploying (so you can find the AdminClient class).

After deployment, start Tomcat and browse at `http://localhost:8080/axis/` to check for the deployment status (3 web services, respectively called TravelAgency, HotelReservation and FlightReservation should be deployed).

5.3.2 web.xml modification

The last step is to modify `axis/WEB-INF/web.xml` file to add description of our demonstration servlets:

- insert into this file the content of `demo/btp-web.xml`, between the last `servlet` and the first `servlet-mapping` tags.

Finally restart Tomcat (to take into account our new servlets).

5.4 Run the demonstration

Start at the url: `http://localhost:8080/axis/btp.html`. This page will give you 3 links

- the Travel Agency Booking, a travel agency client that books a flight and a hotel
- the Travel Agency Viewer, a servlet to see what happens on the participant side (hotel and flight bookings)
- the transaction coordinator admin servlet to monitor the transactions

You can first book a flight and a hotel using the TravelAgencyServlet, then look at what happens at both the transaction coordinator level (SuperiorServlet) and transaction participants level (ParticipantServlet) during the booking process.

To simulate a transaction failure, check the `Transaction failure simulation...` box in the TravelAgencyServlet home page (will cause one of the participants to reject the "prepare" call, then the decider will cancel both participants).

5.5 From CVS

Since all these tasks are really tedious, if you're using CVS version of JOTM-BTP, there is a simpler way to do.
In `jotm-btp/` directory, type:

```
$ ant demo.jar
$ ant demo.copy
```

It will create the `btp-demo.jar` jar files and copy all the needed jars in `axis/WEB-INF/lib/` directory.

Then, you still have to manually insert content of `demo/btp-web.xml` into `axis/WEB-INF/web.xml`.

Start Tomcat.

In `jotm-btp/` directory, type:

```
$ ant demo.deploy
```

That's it!
You can now try the demonstration from `http://localhost:8080/axis/btp.html`.

6 JOTM-BTP architecture

Now that we've seen an example of JOTM-BTP, let see what *is* JOTM-BTP.

6.1 Design goals

At the beginning of JOTM-BTP, the goals were:

- to *Apache Axis* as the target deployment platform, and *JOTM* as the transaction manager
- to provide a *simple* way (as transparent as possible to the end user, i.e. as simple as JTA and XA) to implement and deploy BTP-enabled web services, on both the client and the server side.

- to be *as little intrusive as possible* wrt JOTM: current implementation of JOTM-BTP did not require any modification of JOTM. (JOTM is embedded in some of our web services - those with a coordinator role - as the transaction manager we rely on, and that's it!).

6.1.1 Client-side Design

On the client side, JOTM-BTP provides a simple API to exchange BTP and Application messages with web services. For example, it provides methods to perform `begin()`, `confirm_transaction()`, etc. without having to build, marshal, send, receive and unmarshal XML messages nor to have a fine understanding of BTP messages).

6.1.2 Server-side Design

On the server side, the programmer just has to extend some of our utilitary classes (for example, `Participant` to implement a BTP-enabled participant, which is the equivalent to a `Resource` in JOTM or a `XAResource` in JTA), and to deploy it as a Web Service on Apache Axis.
The programmer has to implement callbacks, that inform him of transaction-related events (like transaction boundaries) - so he doesn't have to cope with BTP, just with his application logic.

6.1.3 BTP implementation design

Java classes are provided for each BTP *Message* (eg. `CONTEXT`, `CONTEXT-REPLY`, `ENROL`, `ENROLLED`,...), as well as *Related Groups* (message compounds, like CONTEXT + APPLICATION).

The main Roles defined in the BTP specification (like *Participant*, *Decider* or *SubCoordinator*) are implemented as java classes, to be deployed as Web Services on Axis. Some embeds a JOTM transaction manager (like `Decider` or `Subcoordinator`, that can create and manage transactions), some don't (like `Participant`, that has some BTP capabilities to be enrolled in a transaction as a resource, but can't create and manage its own transactions).

The transaction-related messages between a coordinator and a participant are undertaken by a *"BTP Resource"* object, enrolled in JOTM as a `Resource`, and exchanging XML messages with the participant it represents under the control of the transaction manager (eg. a `prepare()` call on the resource results in a `PREPARE` BTP message to be sent to the participant, etc.).

6.2 Example: Message flow of CONTEXT + APLLICATION MESSAGE

- The *Terminator* sends a transaction context + application messages to some *Participants*, which are BTP-enabled web services. The `CONTEXT` informs the participants that they have to enrol themselves in a transaction (the `CONTEXT` was obtained from a previous `BEGIN` issued by the terminator).
- The participants extract transaction information from the Context (mainly "superior-address" and "Transaction-Id"), then determine the address of the Coordinator they have to `ENROL` in, and send an `ENROL` message.
- The *Coordinator* has an embedded JOTM transaction manager: upon receipt of the `ENROL` message, it creates a local *Resource* and registers it on JOTM.
- The Resource handles all the XML communication (BTP messages) with the Participant, in reaction to the JOTM transaction manager's decisions.

6.3 Examples

Note: the sample code below does not exactly reflect our current (prototype) APIs, but what we have is very near from what we propose here. Of course, it is subject of changes, and comments are welcome.

6.3.1 Participant code example

```
public class DemoParticipant
   extends org.objectweb.jotm.btp.roles.Participant {

   public void enrolled(String transactionid) {
     System.out.println("Just enrolled in "
                        + transactionid);
   }

   public int prepare(String transactionid) {
     System.out.println("Requested to prepare transaction "
                        + transactionid);
     return Participant.VOTE_PREPARED;
   }
```

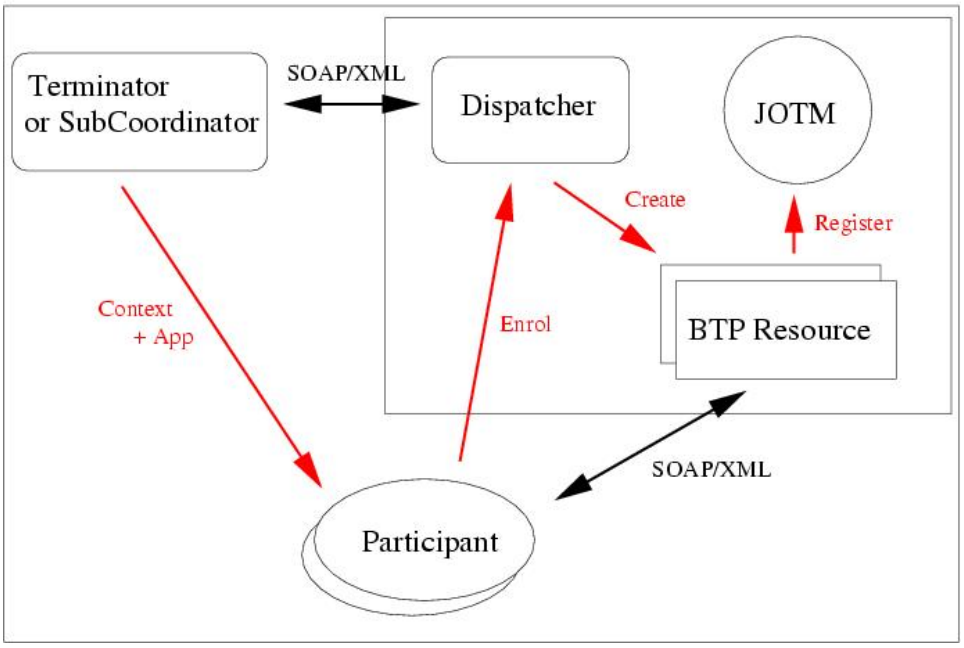

Figure 1: on the left, the terminator; on the right, the coordinator; at the bottom, the participants

```
public boolean confirm(String transactionid) throws Exception {
  System.out.println("Requested to confirm transaction "
                     + transactionid);
  return true;
}

public boolean cancel(String transactionid) throws Exception {
  System.out.println("Requested to cancel transaction "
                     + transactionid);
```

```
    return true;
  }

  public void contradiction(String transactionid) {
  }

  public String applicationMessage(String transactionid,
            org.objectweb.xml.util.XElement message) {
    System.out.println("Got an application message: "
                       + message.toString());
    return "<greetings>Hello World !</greetings>";
  }
}
```

6.3.2 Deployment descriptor for Axis

```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <!-- Demo Participant -->
  <service name="BtpDemoService" provider="java:RPC">
    <parameter name="scope" value="Application"/>
    <parameter name="className" value="DemoParticipant"/>
    <parameter name="allowedMethods" value="messages"/>
  </service>

  <!-- Decider (the transaction manager) -->
  <service name="Decider" provider="java:RPC">
    <parameter name="scope" value="Application"/>
    <parameter name="className"
               value="org.objectweb.jotm.btp.roles.Decider"/>
    <parameter name="allowedMethods" value="messages"/>
  </service>
</deployment>
```

6.3.3 Client code example

```
import  org.objectweb.jotm.btp.client.BtpClient;
import org.objectweb.jotm.btp.messages.*;

 ....

 String deciderAddr =
```

```
   "http://localhost:8080/axis/services/TravelAgency";
String participantAddr =
   "http://localhost:8080/axis/services/HotelReservation";

// Begin the transaction
BtpClient client = new BtpClient();
BtpMessage context = client.begin(deciderAddr);

if(context != null && context.getType() == BtpMessage.CONTEXT) {

  // Build application message
  ApplicationMessage app =
     new ApplicationMessage("<greetings>Hi there !</greetings>");

  // Send it along with the context to the participant
  BtpMessage replyContext =
     client.sendCtxApp(participantAddr, (Context)context, app);

}

// Confirm the transaction
BtpMessage outcome =
   client.confirmTransaction(deciderAddr, (Context)context);

...
```

6.4 XML marshalling and unmarshalling

- Each Java class that represents a BTP message has a `toXml()` method that generates its XML form
- The unmarshalling of messages relies on a `MessageParser` static class: currently based on XmlPull, it relies on a 3rd-party XML parser (XPP) to parse XML, and builds up BTP message classes depending on which BTP message it recognizes in the XML flow.

6.5 Cohesive coordinators (composers)

JOTM does not support *cohesive transactions* (a "cohesive" transaction can be confirmed if a subset of the involved resources - the "confirm set" - answers positively to the `PREPARE` request).
Cohesion is then currently supported by JOTM-BTP using a *hack*: the "BTP Resource" (i.e., the JOTM Resource that is registered in the TM) always confirms the `PREPARE` request for all resources that do not belong to the confirm set.

7 Contacts

If you have some trouble to install JOTM-BTP, any questions or if you want to contribute, do not hesitate to contact us (`mailto:jotm@objectweb.org`).