Getting started with B*lite*C

A tool for rapid development of WS-BPEL applications

May 2010

Powered by Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze - Viale Morgagni, 65 - I-50134 Firenze (ITALY)

Preface

This guide is the primary source of introduction and usage information for B*lite*C.

B*lite*C is a software tool designed for supporting a rapid and easy development of WS-BPEL applications. B*lite*C translates service orchestrations written in B*lite*, a formal language inspired to but simpler than WS-BPEL, into executable WS-BPEL programs.

B*lite*C has been developed as a Master Thesis by Luca Cesari, a student of Computer Science at Università degli Studi di Firenze, under the supervision of Prof. Rosario Pugliese and his collaborators, Alessandro Lapadula and Francesco Tiezzi.

Audience

This guide is intended for users who want to install and use B*lite*C.

Related Documents

For more information about B*lite*, see the following paper

> A. Lapadula, R. Pugliese, and F. Tiezzi. A formal account of WS-BPEL. In *Proc. of 10th international conference on Coordination Models and Languages (COORDINATION)*, volume 5052 of *Lecture Notes in Computer Science*, pages 199215. Springer, 2008. Available at `http://rap.dsi.unifi.it/blite`

For an account of the B*lite* syntax accepted by B*lite*C, see the following paper

> L. Cesari, A. Lapadula, R. Pugliese, and F. Tiezzi. A tool for rapid development of WS-BPEL applications. Tiezzi. In *Proc. 25th ymposium On Applied Computing (SAC)*, SOAP track, pages 2438-2442. ACM, 2010. An extended version is also available at `http://rap.dsi.unifi.it/blite`

The specification document of the OASIS standard Web Services Business Process Execution Language Version 2.0 (WS-BPEL) is available at the following URL

> `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

For more information about ActiveBPEL, an open source implementation of a WS-BPEL engine, you can go directly to the SourceForge page of the project

> `http://sourceforge.net/projects/activebpel502/`

1 Getting and Installing B*lite*C

Download the latest release of B*lite*C from the B*lite*'s Web site

`http://rap.dsi.unifi.it/blite`

Unzip the downloaded zip archive somewhere on your disk, everything needed is inside.

B*lite*C is distributed as a Java Archive (JAR) file and, hence, does not need to be installed in your system. However, since it is a Java program, B*lite*C requires a recent Java Runtime Environment (JRE) to be installed (version 6 or later). If no JRE has previously been installed on your system, you can find one at `http://java.sun.com`.

The outputs produced by B*lite*C are executable WS-BPEL programs. To be executed, these programs must be deployed in a WS-BPEL engine. Currently, the generated WS-BPEL programs are configured for the engine ActiveBPEL.

ActiveBPEL is an open source implementation of a WS-BPEL engine, written in Java. It is made available under the GNU General Public License (GPL) and runs in any standard servlet container. We report here how to install ActiveBPEL:

- Download an ActiveBPEL distribution from `http://sourceforge.net/projects/activebpel502/`. We have tested B*lite*C over version 5.0.2.
- Download a servlet container. We have tested B*lite*C over Apache Jakarta Tomcat version 5.0.28 (available at `http://jakarta.apache.org`). From now on, we consider Apache Jakarta Tomcat as the downloaded servlet container.
- Install Apache Jakarta Tomcat: unzip the downloaded archive and set the environment variables `CATALINA_HOME` and `JAVA_HOME` to the top level directories of your Tomcat and JDK installation, respectively.
- Install ActiveBPEL: unzip the downloaded archive and execute either `install.bat` or `install.sh`, according to your operative system.
- Start the servlet container: execute either `startup.bat` or `startup.sh` (located within the `bin` directory of the Tomcat installation), according to your operative system. To check if the installation properly run, go to `http://XXX:8080/BpelAdmin` by using your favorite browser, where `XXX` is the server's address where Tomcat is running; the ActiveBPEL's administration console should appear (see Figure 1).

To test WS-BPEL applications deployed in the ActiveBPEL engine, you can use any tool for automatic generation of web service requests (i.e. SOAP messages). For this aim, we have used soapUI, which can be freely downloaded from `http://www.soapui.org`. Its installation is straightforward.

Finally, any text editor can be used to write B*lite* programs. Anyway, to simplify the task, we provide users with a customized version of jEdit equipped with specific features supporting programming in B*lite*, such as syntax highlighting, auto indentation and direct compiling. The files for the customization can be downloaded with the B*lite*C distribution archive (see Section 1.1 for the jEdit customization).

We have also implemented a development environment with similar features written in Java. Figure 2 shows a screenshot of our environment. In addition to the functionalities of the customized version of jEdit, our dedicated environment also provides text auto-completion, highlight of search results, local deploy and undeploy. Our development environment already contains the B*lite*C libraries and can be downloaded from the B*lite*'s Web site

`http://rap.dsi.unifi.it/blite`

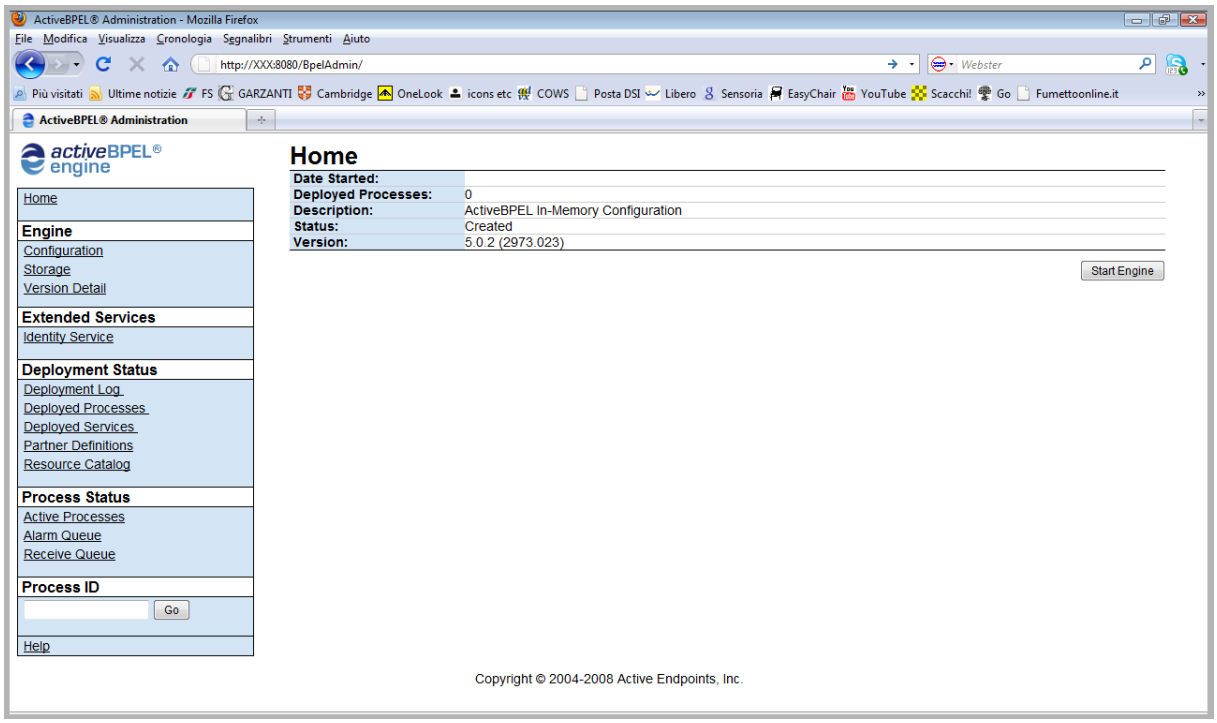

Figure 1: ActiveBPEL administration console

1.1 jEdit customization

jEdit is a programmer's text editor released as free software with full source code, provided under the terms of the GPL 2.0. It is written in Java, so it runs on Mac OS X, OS/2, Unix, VMS and Windows, and has an extensible plugin architecture. We report here how to customize jEdit:

- Download a jEdit distribution from `http://www.jedit.org`. We have used version 4.3.
- Install jEdit; its installation is straightforward.
- Copy `blitec.jar` from the BliteC zip archive to the top level directory of your jEdit installation.
- Copy `blite.xml` from the BliteC zip archive (directory `jedit_files`) to the directory `modes` of your jEdit installation.
- Copy `catalog` from the BliteC zip archive (directory `jedit_files`) to the directory `modes` of your jEdit installation. This operation will cancel any previous customization about editing modalities applied to your jEdit installation. To avoid this, do not copy the file and simply add the following code

  ```
  <MODE NAME="blite" FILE="blite.xml" FILE_NAME_GLOB="*.bl" />
  ```

  within the tag `MODES` of your `catalog` file.
- Copy `blitec.bsh` from the BliteC zip archive (directory `jedit_files`) to the directory `macros\Misc` of your jEdit installation.
- Start jEdit and install the console plugin:
  - select `Plugins` from the menu;

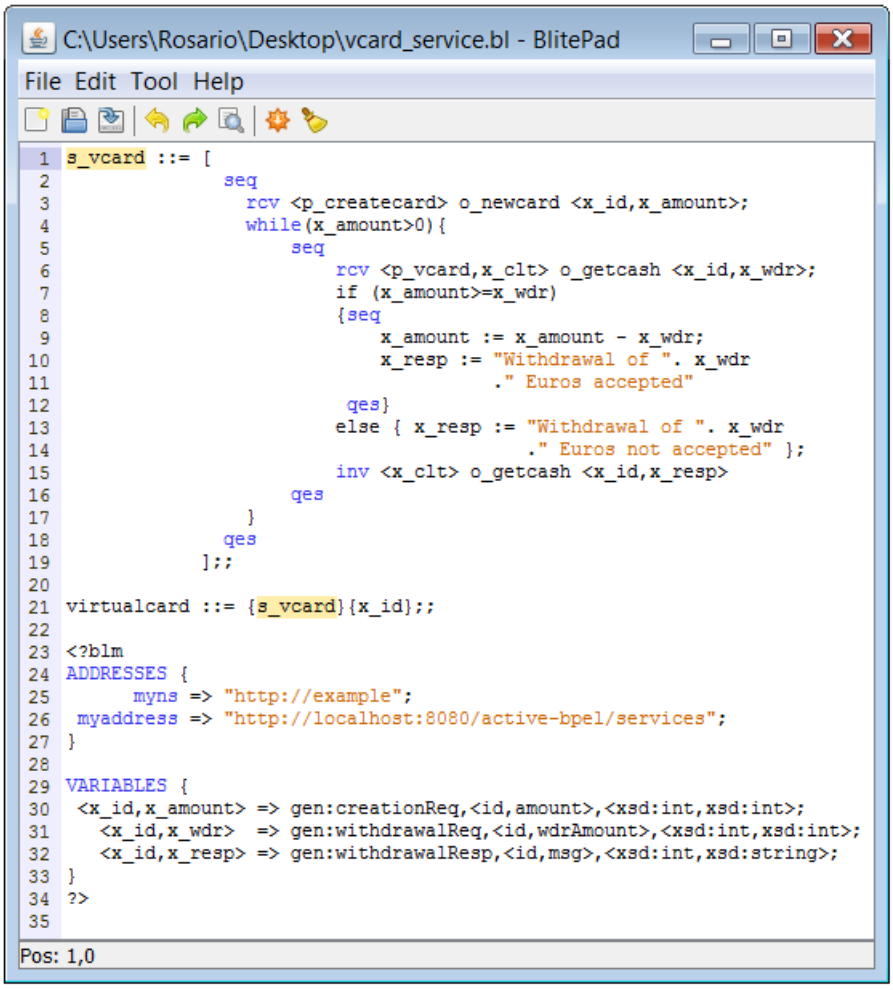

Figure 2: A screenshot of BliteC development environment

  - select `Plugin Manager...`;
  - select the `Install` tab;
  - select the `Console` plugin (we have tested our customization over version 4.4);
  - press the `Install` button.

- Add the BliteC button to the toolbar:
  - select `Utilities` from the menu;
  - select `Global Options...`;
  - select `Tool Bar` on the `JEdit` tree on the left;
  - press the + button on the bottom;
  - select `Command or macro`;
  - select `Macros` on the combo box;
  - select `blitec` on the list;
  - (optionally) specify a built-in icon (e.g. the gear icon corresponding to the image named `22x22/actions/application-run.png`);
  - press the `OK` button twice.

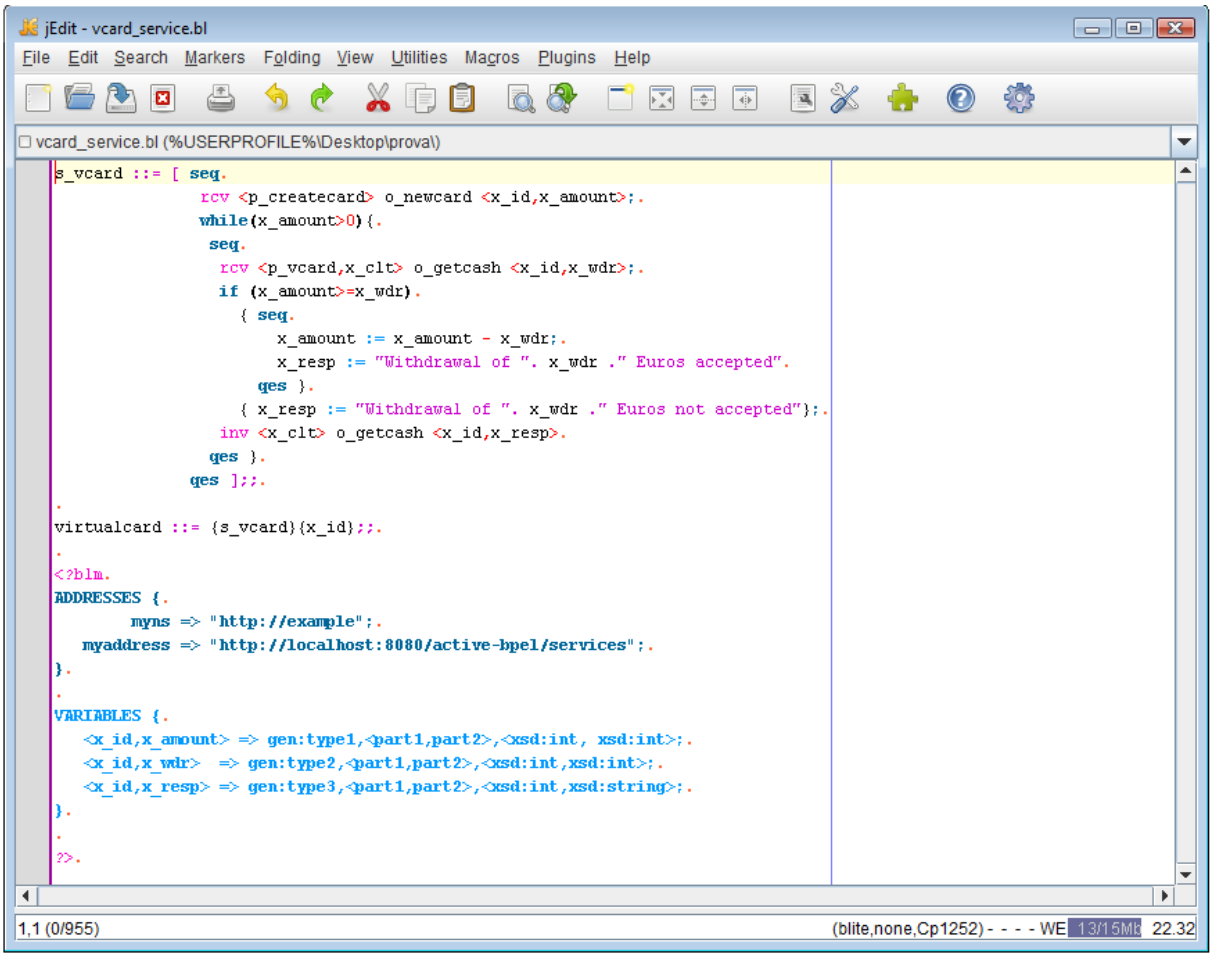

Figure 3: jEdit main window

At the end of the customization, if you open a bl file (you can find some of them within the `example` directory of the zip archive), the jEdit window should look like that in Figure 3. The gear button can be used to execute BliteC to compile the opened Blite program.

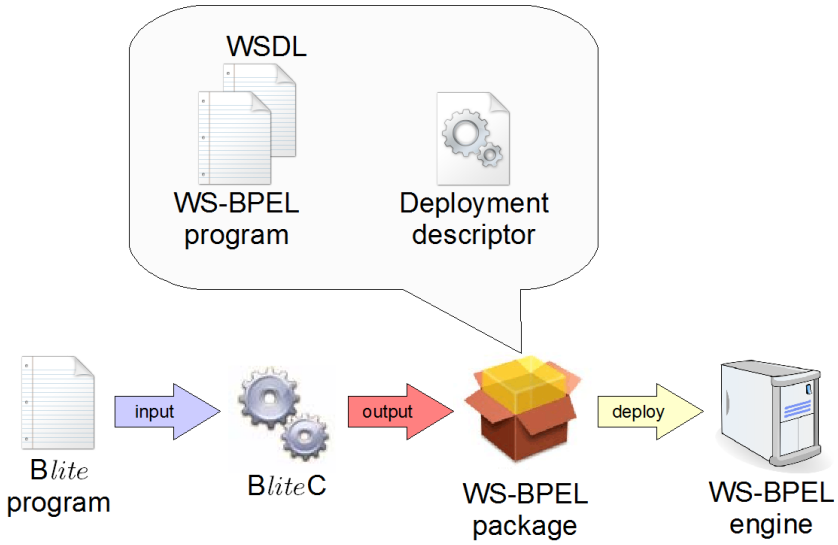

Figure 4: B*lite*C workflow

2 B*lite*C at work

B*lite*C accepts as input a specification written in B*lite* and returns the corresponding WS-BPEL program together with the associated WSDL and deployment descriptor files. The workflow of use of B*lite*C is graphically summarized in Figure 4.

In this section, we describe how to use B*lite*C to design, deploy, and test your first WS-BPEL program. The program is a simple example of a virtual credit card service.

A virtual credit card is a prepaid non-physical credit card devised for safe online shopping. A B*lite* specification for creating and handling a virtual credit card is the following:

```
s_vcard ::=
  [ seq
     rcv <p_createcard> o_newcard <x_id,x_amount>;
     while(x_amount>0){
      seq
       rcv <p_vcard,x_clt> o_getcash <x_id,x_wdr>;
       if (x_amount>=x_wdr)
          { seq
             x_amount := x_amount - x_wdr;
             x_resp := "Withdrawal of ". x_wdr .
                       " Euros accepted"
            qes }
       else { x_resp := "Withdrawal of ". x_wdr .
                        " Euros not accepted"};
       inv <x_clt> o_getcash <x_id,x_resp>
      qes }
    qes ];;

virtualcard ::= {s_vcard}{x_id};;
```

A new card can be created by invoking the operation `o_newcard` by specifying a card identifier and the initial amount. Then, the created instance allows the card holder to perform withdrawals by repeatedly invoking the request-response operation `o_getcash` until the card is empty. For each withdrawal

request, the money availability is checked and a message, stored in `x_resp`, is sent back. The card identifier, stored in `x_id`, is used as a correlation value.

Since the above service does not need to invoke other services, only its address and variables are explicitly declared:

```
<?blm
 ADDRESSES {
   myns => "http://virtualcard";
   myaddress => "http://XXX:8080/active-bpel/services";
 }
 VARIABLES {
   <x_id,x_amount> => gen:creationReq,<id,amount>,
                      <xsd:int,xsd:int>;
     <x_id,x_wdr>  => gen:withdrawalReq,<id,wdrAmount>,
                      <xsd:int,xsd:int>;
    <x_id,x_resp> => gen:withdrawalResp,<id,msg>,
                      <xsd:int,xsd:string>;
 }
?>
```

To compile this Blite program, we have to save the above code into a file (named, e.g., `vcard_service.bl`) and execute the following command[1]:

```
java -jar blite.jar vcard_service.bl
```

This way, the file `virtualcardProcess.bpr`, which is a WS-BPEL package directly deployable into ActiveBPEL, is generated. The WS-BPEL and WSDL files inside the bpr file are as follows.

Virtual card service (WS-BPEL file)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This file is generated by Blite -
    Original Checksum:7b282b7b2e727a61d25551d10e819db4-->
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
         xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:mwl="http://example/virtualcard.wsdl"
         suppressJoinFailure="yes"
         name="virtualcardProcess"
         targetNamespace="http://example/virtualcard.bpel">
  <import location="virtualcard.wsdl"
          namespace="http://example/virtualcard.wsdl"
          importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="cltPL" partnerLinkType="mwl:cltPLT" myRole="p_vcard" />
    <partnerLink name="p_createcardPL"
                 partnerLinkType="mwl:p_createcardPLT" myRole="p_createcard" />
  </partnerLinks>
  <variables>
    <variable name="var1" messageType="mwl:withdrawalReq" />
    <variable name="var2" messageType="mwl:withdrawalResp" />
    <variable name="var0" messageType="mwl:creationReq" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr" properties="mwl:x_idProp" />
```

[1]Here, we assume that files `blitec.jar` and `vcard_service.bl` are located within the same directory. Of course, if it is not the case, relative/absolute paths can be specified as usual.

```
</correlationSets>
<faultHandlers>
  <catchAll>
    <sequence>
      <compensate />
      <empty />
    </sequence>
  </catchAll>
</faultHandlers>
<sequence>
  <sequence>
    <receive partnerLink="p_createcardPL"
             operation="o_newcard" variable="var0" createInstance="yes">
      <correlations>
        <correlation set="x_idCorr" initiate="yes" />
      </correlations>
    </receive>
    <assign>
      <copy>
        <from variable="var0" part="id" />
        <to variable="var1" part="id" />
      </copy>
      <copy>
        <from variable="var0" part="id" />
        <to variable="var2" part="id" />
      </copy>
    </assign>
  </sequence>
  <while>
    <condition>$var0.amount &gt; 0</condition>
    <sequence>
      <receive partnerLink="cltPL" operation="o_getcash" variable="var1">
        <correlations>
          <correlation set="x_idCorr" initiate="no" />
        </correlations>
      </receive>
      <if>
        <condition>$var0.amount &gt;= $var1.wdrAmount</condition>
        <sequence>
          <assign>
            <copy>
              <from>$var0.amount - $var1.wdrAmount</from>
              <to variable="var0" part="amount" />
            </copy>
          </assign>
          <assign>
            <copy>
              <from>concat('Withdrawal of ',string($var1.wdrAmount),
                           ' Euros accepted')</from>
              <to variable="var2" part="msg" />
            </copy>
          </assign>
        </sequence>
        <else>
          <assign>
            <copy>
              <from>concat('Withdrawal of ',string($var1.wdrAmount),
                           ' Euros not accepted')</from>
              <to variable="var2" part="msg" />
            </copy>
          </assign>
        </else>
```

```
        </if>
        <reply operation="o_getcash" partnerLink="cltPL" variable="var2">
          <correlations>
            <correlation set="x_idCorr" initiate="no" />
          </correlations>
        </reply>
      </sequence>
    </while>
  </sequence>
</process>
```

Virtual card service (WSDL file)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This file is generated by Blite -
    Original Checksum:69b5dd70f91f3226f74e179848b380b0-->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:tns="http://example/virtualcard.wsdl"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
                  xmlns:prop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
                  targetNamespace="http://example/virtualcard.wsdl">
  <wsdl:types>
    <xsd:schema targetNamespace="http://example/virtualcard.wsdl">
      <xsd:element name="x_amountEL" type="xsd:int" />
      <xsd:element name="x_idEL" type="xsd:int" />
      <xsd:element name="x_respEL" type="xsd:string" />
      <xsd:element name="x_wdrEL" type="xsd:int" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="creationReq">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="amount" element="tns:x_amountEL" />
  </wsdl:message>
  <wsdl:message name="withdrawalResp">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="msg" element="tns:x_respEL" />
  </wsdl:message>
  <wsdl:message name="withdrawalReq">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="wdrAmount" element="tns:x_wdrEL" />
  </wsdl:message>
  <wsdl:portType name="p_createcardPT">
    <wsdl:operation name="o_newcard">
      <wsdl:input message="tns:creationReq" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="p_vcardPT">
    <wsdl:operation name="o_getcash">
      <wsdl:input message="tns:withdrawalReq" />
      <wsdl:output message="tns:withdrawalResp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="p_createcardBinding" type="tns:p_createcardPT">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="o_newcard">
      <soap:operation
           soapAction="http://example/virtualcard.wsdl/o_newcard" style="document" />
      <wsdl:input>
```

```
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="p_vcardBinding" type="tns:p_vcardPT">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="o_getcash">
      <soap:operation
           soapAction="http://example/virtualcard.wsdl/o_getcash" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="p_createcardService">
    <wsdl:port name="p_createcardPort" binding="tns:p_createcardBinding">
      <soap:address
           location="http://localhost:8080/active-bpel/services/p_createcardService" />
    </wsdl:port>
  </wsdl:service>
  <wsdl:service name="p_vcardService">
    <wsdl:port name="p_vcardPort" binding="tns:p_vcardBinding">
      <soap:address
           location="http://localhost:8080/active-bpel/services/p_vcardService" />
    </wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="p_createcardPLT">
    <plnk:role name="p_createcard" portType="tns:p_createcardPT" />
  </plnk:partnerLinkType>
  <plnk:partnerLinkType name="cltPLT">
    <plnk:role name="p_vcard" portType="tns:p_vcardPT" />
  </plnk:partnerLinkType>
  <prop:property name="x_idProp" type="xsd:int" />
  <prop:propertyAlias propertyName="tns:x_idProp"
                      messageType="tns:withdrawalResp" part="id" />
  <prop:propertyAlias propertyName="tns:x_idProp"
                      messageType="tns:withdrawalReq" part="id" />
  <prop:propertyAlias propertyName="tns:x_idProp"
                      messageType="tns:creationReq" part="id" />
</wsdl:definitions>
```

Notice that a WSDL file, which is a copy of that inside the bpr file, is generated together with the bpr file. This permits supporting the chained execution of B*lite*C over a set of B*lite* programs with dependencies (i.e. some program of the set imports some WSDL definitions from other programs of the set). See, e.g., the B*lite* programs within the directory `examples\shipping`.

To deploy the bpr file, it is sufficient to move it into the engine's deployment directory `bpr`. Then, to check that the deploy succeeded, we can use the ActiveBPEL's administration console that can be accessed by using any browser at the address `http://XXX:8080/BpelAdmin` (where `XXX` is the server's address where the ActiveBPEL engine is running). By selecting `Deployed Processes` from the menu on the left-hand side, we obtain the list of the deployed processes (Figure 5) among which `virtualcardProcess` should appear. Now, by selecting `Deployed services`, we can retrieve the URLs of the two WSDL files corresponding to the two partner links for interacting with the service:

```
http://XXX:8080/active-bpel/services/p_createcardService?wsdl
http://XXX:8080/active-bpel/services/p_vcardService?wsdl
```

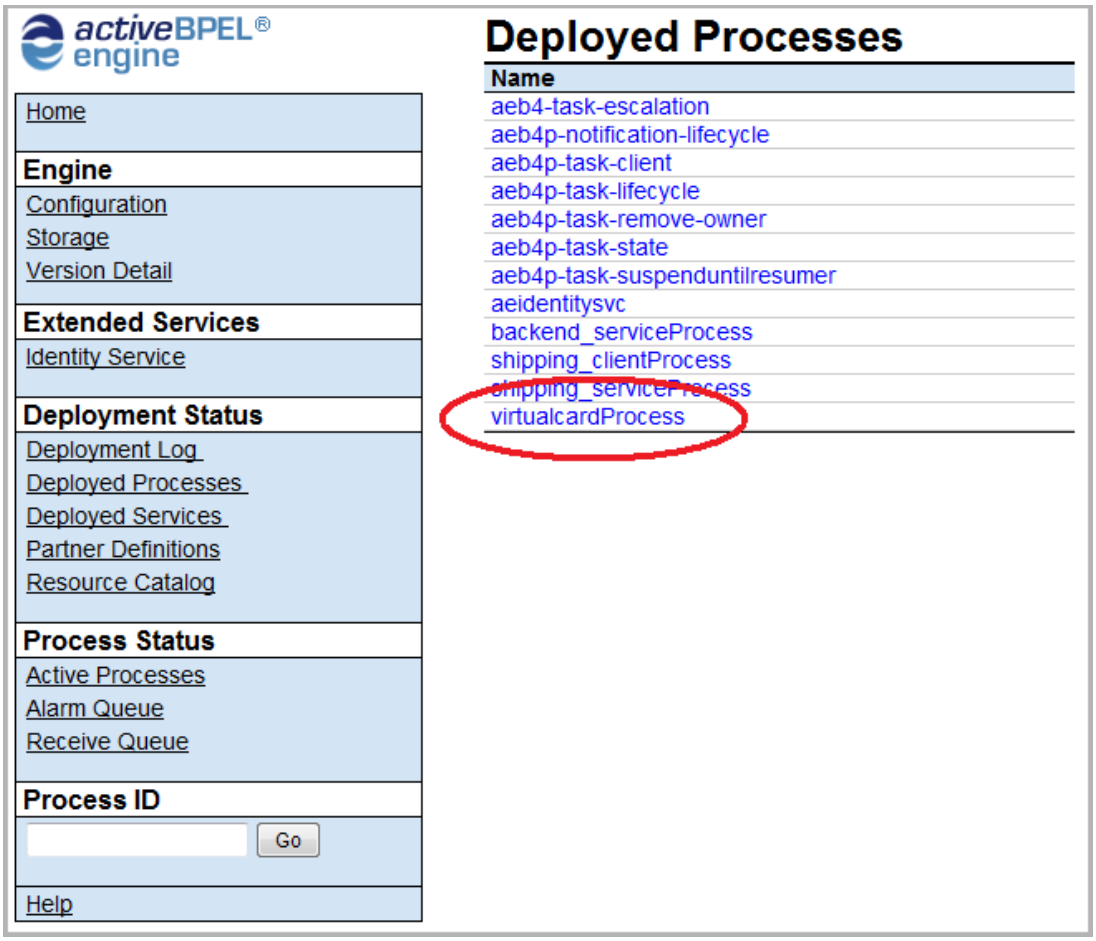

Figure 5: List of deployed processes

Indeed, ActiveBPEL splits the WSDL file contained in the bpr package in as many WSDL files as the number of partner links used by the business process.

Finally, by using soapUI, we can invoke the service by sending the following SOAP messages:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:vir="http://virtualcard/virtualcard.wsdl">
    <soapenv:Header/>
    <soapenv:Body>
        <vir:x_idEL> 1234 </vir:x_idEL>
        <vir:x_amountEL> 100 </vir:x_amountEL>
    </soapenv:Body>
</soapenv:Envelope>

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:vir="http://virtualcard/virtualcard.wsdl">
    <soapenv:Header/>
    <soapenv:Body>
        <vir:x_idEL> 1234 </vir:x_idEL>
        <vir:x_wdrEL> 50 </vir:x_wdrEL>
    </soapenv:Body>
</soapenv:Envelope>
```

The first message creates a virtual credit card identified by `1234` with `100` Euros as initial amount, while the second message is a request for withdrawing `50` Euros. In response to the second message we get the string `Withdrawal of 50 Euros accepted` and, by selecting `Active Processes` from the console menu, we can verify that the card instance is still running. If we resend the withdrawal request we obtain the same response, but the instance status changes to `Completed`. Instead, if we send the following message as third message:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:vir="http://virtualcard/virtualcard.wsdl">
    <soapenv:Header/>
    <soapenv:Body>
        <vir:x_idEL> 1234 </vir:x_idEL>
        <vir:x_wdrEL> 80 </vir:x_wdrEL>
    </soapenv:Body>
</soapenv:Envelope>
```

we get in response the string `Withdrawal of 80 Euros not accepted` and the status of the card instance remains `Running`.