

# A tool for rapid development of WS-BPEL applications

Luca Cesari, Rosario Pugliese and Francesco Tiezzi  
Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze  
cesari.luca@gmail.com, pugliese@unifi.it, tiezzi@dsi.unifi.it

## ABSTRACT

WS-BPEL is imposing itself as a standard for orchestration of web services. However, there are still some well-known difficulties that make programming in WS-BPEL a tricky task. In this paper, we present *BliteC*, a software tool we have developed for supporting a rapid and easy development of WS-BPEL applications. *BliteC* translates service orchestrations written in *Blite*, a formal language inspired to but simpler than WS-BPEL, into readily executable WS-BPEL programs. We illustrate our approach by means of a few practical programming examples.

## Categories and Subject Descriptors

D.2.2 [Software engineering]: Design Tools and Techniques—*Computer-aided software engineering*; D.3.1 [Programming Languages]: Formal Definitions and Theory—*Syntax Semantics*; D.3.4 [Programming Languages]: Processors—*Compilers Parsing*

## Keywords

Service-Oriented Computing, Web services, Compilers

## 1. INTRODUCTION

In recent years, there has been an ever increasing acceptance of WS-BPEL [29] as a standard language for orchestration of *web services*, one of the most successful and well-developed implementations of the *Service-Oriented Computing (SOC)* paradigm. However, designing and developing WS-BPEL applications is a difficult and error-prone task. The language has an XML syntax which makes awkward writing WS-BPEL code by using standard editors. Therefore, many companies (among which e.g. Oracle and Active Endpoints) have equipped their WS-BPEL engines with graphical designers. Such tools are certainly suitable to develop simple business processes, but turn out to be cumbersome and ineffective when programming with more complex applications. Further difficulties derive from the fact that WS-BPEL is equipped with such intricate features as concurrency, mul-

tiplex service instances, message correlation, long-running business transactions, termination and compensation handlers. Most of all, WS-BPEL comes without a formal semantics and its specification document, written in ‘natural’ language, contains a fair number of acknowledged ambiguous features that may give rise to different interpretations. These ambiguities have led to engines implementing different semantics (see [24]) and, hence, have undermined portability of WS-BPEL programs across different platforms. Therefore, many research efforts have been devoted to provide WS-BPEL with a formal semantic (see, e.g., [26, 17, 32, 30, 23, 25, 19, 20]), although most of them do not deal with the complete language. Finally, the deployment procedure of WS-BPEL programs is not standardised, which further compromises portability. In fact, to execute a WS-BPEL program, besides the associated WSDL [16] document that describes the program’s public interfaces, different engines require different (and not integrable) *process deployment descriptors*, i.e. sets of configuration files that describe how the program should be deployed.

To overcome these difficulties, we have developed *BliteC*, a software tool that accepts as an input a specification written in the lightweight orchestration language *Blite* [24] and returns the corresponding WS-BPEL program together with the associated WSDL and deployment descriptor files.

*Blite* is closely inspired to WS-BPEL. It is the result of a tension between handiness and expressiveness. While the set of WS-BPEL constructs is not intended to be a minimal one, the design of *Blite*, to keep the language manageable, only retains the core features of WS-BPEL. It follows that *Blite* is simpler and more compact than WS-BPEL, although it maintains the same descriptive power. Using *Blite* for initially specifying a service orchestration offers some significant advantages. From the one hand, *Blite* textual notation is certainly more manageable than those, possibly graphical, notations proposed for WS-BPEL. From the other hand, *Blite* is equipped with a formal operational semantics that clarifies all ambiguous and intricate aspects of WS-BPEL. Of course, to preserve such semantics on different WS-BPEL engines, the translation of *Blite* programs into WS-BPEL ones has to be properly targeted to each specific engine.

*BliteC* further simplifies the programmers work by automating the deployment procedure. In fact, the returned files are properly packaged to be immediately executable in a WS-BPEL engine. Currently, these packages are intended

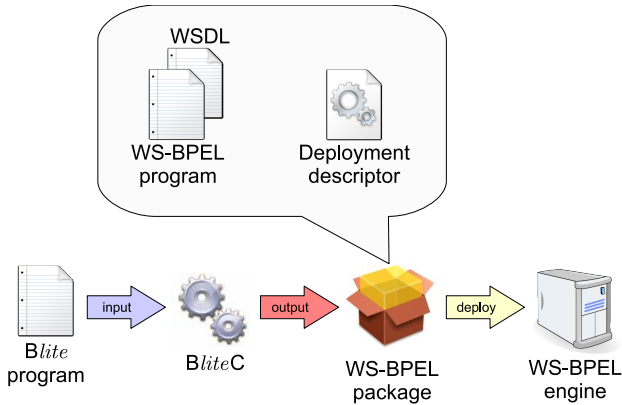


Figure 1: **BliteC** workflow

to be deployed on ActiveBPEL [6] that, according to [24], is one of the freely available WS-BPEL engines that better complies with the WS-BPEL specification. The workflow of use of *BliteC* is graphically depicted in Figure 1.

The rest of the paper is organized as follows. Section 2 summarizes WS-BPEL, while Section 3 introduces the syntax of *Blite* accepted by our tool. Section 4 presents *BliteC* and explains the correspondence between *Blite* constructs and WS-BPEL activities. Section 5 illustrates *BliteC* at work on some practical examples, two of which are borrowed from the official specification of WS-BPEL. Section 6 reviews more closely related work and hints at some future work.

## 2. AN OVERVIEW OF WS-BPEL

WS-BPEL is essentially a linguistic layer on top of WSDL for describing the structural aspects of web service orchestration. In WS-BPEL, the logic of interaction between a service and its environment is described in terms of structured patterns of communication actions composed by means of control flow constructs that enable the representation of complex structures. Orchestration exploits state information that is maintained through shared variables and managed through message correlation. For the specification of orchestration, WS-BPEL provides many different *basic* and *structured activities*.

The following basic activities are provided: `<receive>` and `<reply>`, to enable web service one-way and request-response operations; `<invoke>`, to invoke web service operations; `<wait>`, to delay execution for some amount of time; `<assign>`, to update the values of variables with new data; `<throw>`, to signal internal faults; `<exit>`, to immediately end a service instance; `<empty>`, to do nothing; `<compensate>` and `<compensateScope>`, to invoke compensation handlers; `<rethrow>`, to propagate faults; `<validate>`, to validate variables; and `<extensionActivity>`, to add new activity types.

The structured activities describe the control flow logic of a business process by composing basic and/or structured activities recursively. The following structured activities are provided: `<sequence>`, to execute activities sequentially; `<if>`, to execute activities conditionally; `<while>` and `<repeatUntil>`, to repetitively execute activities; `<flow>`, to execute activities in parallel; `<pick>`, to execute activities selectively; `<forEach>`, to (sequentially or in parallel) execute multiple activities;

and `<scope>`, to associate handlers for exceptional events to a primary activity.

Notably, synchronization dependencies among activities, other than by means of control flow constructs, can also be specified through *flow links* to form directed acyclic graphs. A flow link is a conditional transition that connects a ‘source’ activity to a ‘target’ activity. When a source activity completes, the associated *transition condition* is evaluated to determine the status of the *join condition* that acts on the flow link of the target activity. A target activity may only start when all its source activities complete and its join condition evaluates to true.

The handlers within a `<scope>` can be of four different kinds: `<faultHandler>`, to provide the activities in response to faults occurring during execution of the primary activity; `<compensationHandler>`, to provide the activities to compensate the successfully executed primary activity; `<terminationHandler>`, to control the forced termination of the primary activity; and `<eventHandler>`, to process message or timeout events occurring during execution of the primary activity. If a fault occurs during execution of a primary activity, the control is transferred to the corresponding fault handler and all currently running activities inside the scope are interrupted immediately without involving any fault/compensation handling behaviour. If another fault occurs during a fault/compensation handling, then it is re-thrown, possibly, to the immediately enclosing scope. Compensation handlers attempt to reverse the effects of previously successfully completed primary activities (scopes) and have been introduced to support Long-Running (Business) Transactions (LRTs). Compensation can only be invoked from within fault or compensation handlers starting the compensation either of a specific inner (completed) scope, or of all inner completed scopes in the reverse order of completion. The latter alternative is also called the *default* compensation behaviour. Invoking a compensation handler that is unavailable is equivalent to perform an empty activity.

A WS-BPEL program, also called (*business*) *process*, is a `<process>`, that is a sort of `<scope>` without compensation and termination handlers.

WS-BPEL uses the basic notion of *partner link* to directly model peer-to-peer relationships between services. This relationship is expressed at the WSDL level by specifying the roles played by each of the services in the interaction. However, the information provided by partner links is not enough to deliver messages to a business process. Indeed, since multiple instances of a same service can be simultaneously active because service operations can be independently invoked by several clients, messages need to be delivered not only to the correct partner, but also to the correct instance of the service that the partner provides. To achieve this, WS-BPEL relies on the business data exchanged rather than on specific mechanisms, such as *WS-Addressing* [21] or low-level methods based on SOAP headers. In fact, WS-BPEL exploits *correlation sets*, namely sets of *correlation variables* (called *properties* in WS-BPEL jargon), to declare the parts of a message that can be used to identify an instance. This way, a message can be delivered to the correct instance on the basis of the values associated to the correlation variables,

independently of any routing mechanism.

We end this section by showing an auction service described in the official specification of WS-BPEL [29, Sect. 15.4]. This example will allow us to illustrate most of the language features, including correlation sets, shared variables, control flow structures, asynchronous communication and multiple start activities, and, through its implementation in *Blite* presented in Section 5.3, will permit a rough comparison between the two languages.

The auction service collects information from a seller and a buyer about a concluded auction, reports the auction result to an auction registration service, and then communicates the registration result to the seller and the buyer. The auction house process may be instantiated either by receiving the seller information or by receiving the buyer information. Indeed, the process is able of receiving the seller and buyer requests in a statically unpredictable order and in such a way that the first incoming message triggers the creation of a process instance which the subsequent request is delivered to. This requires the two starting receive activities to share a correlation set, which will be initialized with an auction identifier that the seller and the buyer have to provide when sending their requests. The auction house process passes the auction identifier to the auction registration service that, in its turn, returns the identifier in its answer to locate the proper process instance.

We report below the corresponding WS-BPEL program, where to make the reading of the code easier, we have omitted irrelevant details<sup>1</sup> and highlighted the basic activities receive, invoke and assign.

```
<process name="auctionService" ... >
<partnerLinks> ... </partnerLinks>
<variables> ... </variables>
<correlationSets>
  <correlationSet name="auctionIdentification"
    properties="as:auctionId" />
</correlationSets>
<sequence>
  <flow>
    <receive name="acceptSellerInformation"
      partnerLink="seller"
      portType="as:sellerPT"
      operation="submit"
      variable="sellerData"
      createInstance="yes">
      <correlations>
        <correlation set="auctionIdentification"
          initiate="join" />
      </correlations>
    </receive>
    <receive name="acceptBuyerInformation"
      partnerLink="buyer"
      portType="as:buyerPT"
      operation="submit"
      variable="buyerData"
      createInstance="yes">
      <correlations>
        <correlation set="auctionIdentification"
          initiate="join" />
      </correlations>
    </receive>
  </flow>
  <assign>
    <copy>
      <from>
        ... http://example.com/auction/RegistrationService ...
      </from>
      <to partnerLink="auctionRegistrationService" />
    </copy>
```

<sup>1</sup>The fully detailed version of the WS-BPEL process and the associated WSDL document can be found in [29, Sect. 15.4].

```
<copy>
  <from partnerLink="auctionRegistrationService"
    endpointReference="myRole" />
  <to>${auctionData.auctionHouseEndpointReference}</to>
</copy>
<copy>
  <from>${sellerData.auctionId}</from>
  <to>${auctionData.auctionId}</to>
</copy>
<copy>
  <from>1</from>
  <to>${auctionData.amount}</to>
</copy>
</assign>

<invoke name="registerAuctionResults"
  partnerLink="auctionRegistrationService"
  portType="as:auctionRegistrationPT"
  operation="process"
  inputVariable="auctionData" />

<receive name="receiveAuctionRegistrationInformation"
  partnerLink="auctionRegistrationService"
  portType="as:auctionRegistrationAnswerPT"
  operation="answer"
  variable="auctionAnswerData">
  <correlations>
    <correlation set="auctionIdentification" />
  </correlations>
</receive>

<flow>
  <sequence>
    <assign>
      <copy>
        <from>${sellerData.endpointReference}</from>
        <to partnerLink="seller" />
      </copy>
      <copy>
        <from><literal>Thank you!</literal></from>
        <to>${sellerAnswerData.thankYouText}</to>
      </copy>
    </assign>

    <invoke name="respondToSeller"
      partnerLink="seller"
      portType="as:sellerAnswerPT"
      operation="answer"
      inputVariable="sellerAnswerData" />

  </sequence>
</flow>
</sequence>
</process>

<assign>
  <copy>
    <from>${buyerData.endpointReference}</from>
    <to partnerLink="buyer" />
  </copy>
  <copy>
    <from><literal>Thank you!</literal></from>
    <to>${buyerAnswerData.thankYouText}</to>
  </copy>
</assign>

<invoke name="respondToBuyer"
  partnerLink="buyer"
  portType="as:buyerAnswerPT"
  operation="answer"
  inputVariable="buyerAnswerData" />

</sequence>
</flow>
</sequence>
</process>
```

Notice that the buyer and the seller provide their endpoint references for the auction house process to respond properly in an asynchronous way. For similar reasons, the auction house process provides its own endpoint reference to the auction registration service.

### 3. PROGRAMMING SERVICES IN BLITE

A *Blite* program accepted by *BliteC* is composed of a *Blite* specification and a declarative part. The former focusses on the behavioural aspects of the orchestration, while the latter provides the implementation details (e.g. types, addresses, bindings, ...) that are necessary to deploy and execute the corresponding WS-BPEL program.

### 3.1 Blite specification

*Blite* [24] is a prototypical orchestration language whose design is closely inspired to WS-BPEL. To keep the language manageable, the design of *Blite* only retains the core features of WS-BPEL. In fact, some aspects have been intentionally left out, including timeouts, synchronization dependencies within flow activities, event and termination handlers. Moreover, *Blite* only provides a simplified form of fault and compensation handling and only supports unnamed faults and the default compensation mechanisms.

*Blite* provides a formal description of service deployments by only keeping relevant implementation details. Thus, the roles played by service partners in a service interaction are explicitly indicated by *partner links* and *partners*, while such aspects as physical *service binding* (necessary to generate the associated WSDL documents and deployment descriptors) are abstracted away and dealt with separately in the declarative part.

The syntax of *Blite* accepted by *BliteC* is given in Figure 2. Services are *structured activities* built from *basic activities*, i.e. service invocation, service request processing, assignment, empty activity, fault generation and instance forced termination, by exploiting operators for conditional choice, iteration, sequential composition, parallel composition, pick and scope. A scope activity groups a primary activity  $A$  together with a fault handling activity  $A_f$  and a compensation activity  $A_c$ . *Start activities* are structured activities that initially can only execute receive activities. Sequence has higher priority (i.e. bind more tightly) than parallel composition and pick. Moreover, fault and compensation activities may be omitted from a scope construct, in which case they are intended to be *throw* and *empty*, respectively.

Notation  $\langle \cdot \rangle$  stands for tuples of objects, e.g.  $\langle x_1, \dots, x_n \rangle$  denotes a tuple of variables (variables in the same tuple must be pairwise distinct). Partner links  $pl$  can be either of the form  $\langle partner \rangle$  or of the form  $\langle partner_1, partner_2 \rangle$ . Indeed, in one-way interactions a partner link indicates a single partner because one of the parties provides all the invoked operations. Instead, in asynchronous request-response interactions, partner links indicate two partners because the requesting partner must provide a callback operation used by the receiving partner to send notifications. Service partners used for receiving messages must be known at design-time, while the partners used to send messages in reply may be dynamically determined.

Besides asynchronous invocation, WS-BPEL also provides a construct for synchronous invocation of remote services. This construct forces the invoker to wait for an answer by the invoked service, that indeed performs a pair of activities *receive-reply*. In *Blite*, this behaviour is rendered in terms of a pair of activities *invoke-receive* over the same operation executed by the invoker and a corresponding pair of activities *receive-invoke* executed by the invoked service.

Data can be shared among different activities through *shared variables* (ranged over by  $x, x_1, \dots$ ). The manipulable values are boolean, integer numbers (ranged over by *int*), strings (as usual, written within double inverted commas), partner links, and literals (defined in the declarative part

and denoted by putting the symbol  $\$$  in front of the corresponding identifier). *Expressions* combine values and variables by means of boolean, arithmetic, comparison and string operators. Operators  $set(x, "path")$  and  $get(x, "path")$  can be used respectively in the left and right hand sides of an assignment to act on a specific element (indicated by *path*) of the XML message stored in the variable  $x$ . Both operators turn out to be quite useful for easily interacting with non-*Blite* services (see Section 5.6).

*Blite* specifications are finite compositions of *definitions* (that assign names to *Blite* terms), containing at most one *deployment* definition. A deployment associates a *correlation set*, namely a (possibly empty) set of correlation variables, to a *service*. A service provides a ‘top-level’ scope (i.e. a scope that cannot be compensated) and offers a choice of alternative receives among multiple start activities.

We refer the interested reader to [24] for a formal account of the *Blite* operational semantics.

### 3.2 Declarative part

The declarative part of a *Blite* program specifies configuration data necessary to properly translate the *Blite* specification into an executable WS-BPEL program. Notably, *BliteC* requires the user to insert only the strictly necessary data. The declarations must be included within  $\langle ?blm \rangle$  and  $\langle ? \rangle$ , and can occur in any position within a *Blite* program.

A declarative part has the following form:

```
<?blm
  ADDRESSES {
    myns => "base_for_namespaces";
    myaddress => "base_for_service_url";
  }
  IMPORTS {
    associations prefix => "url";
  }
  VARIABLES {
    variable and message declarations
  }
  LITERALS {
    associations literal_name => [[ literal_code ]];
  }
  PARTNERLINKS {
    partner link type declarations
  }
?>
```

where blocks `ADDRESSES` and `VARIABLES` are mandatory, while the other ones can be omitted.

Within the `ADDRESSES` block the user has to specify the base for the namespaces used inside the generated files (after the keyword `myns`) and the base for the address where the new service will be hosted (after the keyword `myaddress`).

To define a service orchestration it is often necessary to import data (e.g. type declarations) from documents (e.g. WSDL files) associated to other services. To this aim, the user can specify the addresses of the documents to be imported within the `IMPORTS` block, by associating to each imported document a namespace prefix that will be used in the subsequent declarations to refer to it. Notably, definitions belonging to standard namespaces (e.g. <http://www.w3.org/2001/XMLSchema>) are automatically imported and, hence, do not require any declaration.

*Blite* variables are untyped, while WS-BPEL ones must be



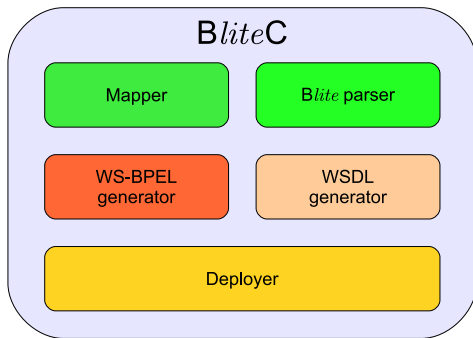


Figure 3: *BliteC* architecture

## 4.1 *BliteC* architecture

*BliteC*<sup>2</sup> is developed in Java<sup>3</sup> to guarantee its portability across different platforms, to exploit the well-established libraries for generating parsers and for manipulating XML documents, and because Java is the reference language for the applications designed around WS-BPEL. Besides the standard Java libraries, we have used JDOM [12] for creating and managing XML documents, JavaCC [11] for generating the parsers that validate the input documents, and JJTree<sup>4</sup> for allowing the parsers to build parse trees (already arranged to support the Visitor design pattern [18]).

The architecture of *BliteC* is graphically depicted in Figure 3. The tool is composed of five main components:

- *Mapper* parses the declarative part of the input *Blite* program and initializes a map that associates each declared object (e.g. partner link, literal, variable, ...) to its name;
- *Blite parser* analyzes the *Blite* specification within the input program, completes the map created by *Mapper* and creates the parse tree of the *Blite* specification;
- WS-BPEL and WSDL *generators* use the data produced by the above components to generate a WS-BPEL process and the associated WSDL document;
- *Deployer* generates the deployment descriptor and packages all created documents into a deployable file; it is the only ‘engine-dependent’ component.

Any text editor can be used to write *Blite* programs. Anyway, to simplify the task, we provide users with a customized version of jEdit<sup>5</sup> [2] equipped with specific features supporting programming in *Blite*, such as syntax highlighting, auto indentation and direct compiling. The files for the customization can be downloaded with the *BliteC* distribution archive. The main advantage of jEdit with respect to more professional development environments is that it is multi-platform and lightweight. We are also implementing a development environment with similar features written in Java. Figure 4 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.

<sup>2</sup>*BliteC* is a free software; it can be downloaded from <http://rap.dsi.unifi.it/blite> and redistributed and/or modified under the terms of the GNU General Public License.

<sup>3</sup>JRE and JDK version 6.

<sup>4</sup>JJTree is included within JavaCC.

<sup>5</sup>jEdit is a programmer’s text editor written in Java released

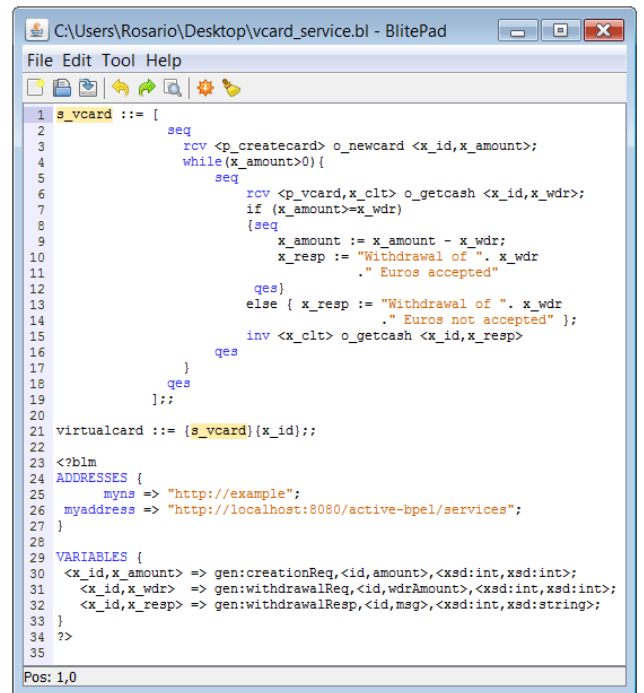


Figure 4: *BliteC* development environment

## 4.2 From *Blite* to WS-BPEL

We now provide some insights about the transformation of *Blite* constructs into WS-BPEL activities. Since the same WS-BPEL program might have different behaviours on different engines [24], the translation described here is targeted to a specific engine, i.e. ActiveBPEL. If one want to produce packages intended to be executed by other WS-BPEL engines, the translation has possibly to be properly tailored. Since there is no precise description of the behaviour of the ActiveBPEL engine, it cannot be formally proved that the semantics of the WS-BPEL program resulting from a translation conforms to that of the original *Blite* program. However, since *Blite* is a ‘sort of’ lightweight variant of WS-BPEL, the translation we define is quite intuitive and direct, which makes us confident that the original semantics is obeyed. This is of course witnessed by all the experiments we have done.

Communication activities, invokes and receives, are translated in a different way depending on their arguments and their position in the code. Therefore, the translation of *Blite* programs proceeds in a top-down fashion and, in doing so, the WS-BPEL generator exploits the information previously collected by the *Mapper* and the *Blite* parser. For example, as shown in Table 1, if a receive activity is positioned within a *pcx* construct it is translated as an *<onMessage>* activity; if it is positioned after an *invoke* (in case of a request-response interaction) it is translated as a synchronous *<invoke>*; otherwise, it is simply translated as a *<receive>*. In addition to the excerpts shown in the table, the translation also settles the following activity attributes. If a receive is a start activity, to allow the process to be instantiated, the *createInstance*

as free software with full source code, provided under the terms of the GPL 2.0.

**Table 1: Mapping of the receive activity**

<i>Blite</i>	WS-BPEL
pck ... rcv <i>pl</i> op <x1,...,xn>... kcp	<onMessage partnerLink="pl" operation="op" variable="x" />
inv <p,p'> op <y1,...,yn>; rcv <p'> op <x1,...,xm>	<invoke partnerLink="pl" operation="op" inputVariable="y" outputVariable="x" />
rcv <i>pl</i> op <x1,...,xn>	<receive partnerLink="pl" operation="op" variable="x" />

**Table 2: Mapping of the invoke activity**

<i>Blite</i>	WS-BPEL
inv <i>pl</i> op <x1,...,xn>	<invoke partnerLink="pl" operation="op" variable="x" />
inv <p,p'> op <y1,...,yn>; rcv <p'> op <x1,...,xm>	<invoke partnerLink="pl" operation="op" inputVariable="y" outputVariable="x" />
rcv <p,p'> op <y1,...,yn> ... inv <p'> op <x1,...,xm>	<receive ... /> ... <reply partnerLink="pl" operation="op" variable="x" />

attribute must be set to *yes*. Moreover, if some correlation variables are involved, the corresponding correlation set (whose declaration is generated during the translation of the deployment term) must be specified as a further argument of the <receive> activity. The correlation attributes *initiate* and *pattern* are set according to the type of the interaction.

The invoke activity is translated similarly, as shown in Table 2; in particular, when it is used in a request-response interaction to send the response, it is translated as a <reply> activity. The translation of the remaining basic activities, as shown in Table 3, is straightforward. In particular, an assign activity involving message variables is translated by possibly using XPATH queries and by exploiting the type of the involved variables (defined in the declarative part) to identify the corresponding parts. Also the translation of the structured activities does not require a significant effort, as shown in Table 4. Finally, as shown in Table 5, a *Blite* service is rendered as a scope, where the compensation handler is removed and the tag <scope> is replaced by <process>.

## 5. BLITEC AT WORK

In this section, we present an application of *BliteC* to some illustrative practical scenarios. The WS-BPEL and WSDL files of the presented services are reported in the Appendix, while all *Blite* programs and the corresponding WS-BPEL packages can be retrieved along with the *BliteC* distribution archive.

### 5.1 A virtual credit card service

A virtual credit card is a prepaid non-physical credit card devised for safe online shopping. A *Blite* 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)
```

**Table 3: Mapping of assign, empty, throw and exit**

<i>Blite</i>	WS-BPEL
<i>x</i> := <i>e</i>	<assign> <copy> <from> <i>e</i> /<from> <to> \$var_x.part_x /<to> </copy> </assign>
set( <i>x</i> , "path") := <i>e</i>	<assign> <copy> <from> <i>e</i> /<from> <to variable="var_x" part="part_x"> <query> path /<query> </to> </copy> </assign>
<i>y</i> := get( <i>x</i> , "path")	<assign> <copy> <from variable="var_x" part="part_x"> <query> path /<query> </from> <to variable="var_y" part="part_y" /> </copy> </assign>
empty	<empty />
throw	<throw />
exit	<exit />

```
{ 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 is created by invoking the operation *o\_newcard* and specifying a card identifier and the initial amount. 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 fact that the invoke activity used for the reply is performed along the same operation of the second receive indicates that the two activities form a synchronous request-response interaction, hence the invoke will be translated into a <reply> activity. 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 command `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. Of course, the same actions can be also performed by using the editor

Table 4: Mapping of structured activities

<i>Blite</i>	WS-BPEL
if (e) {a <sub>1</sub> } else {a <sub>2</sub> }	<pre>&lt;if&gt;   &lt;condition&gt; e &lt;/condition&gt;   a<sub>1</sub> &lt;else&gt; a<sub>2</sub> &lt;/else&gt; &lt;/if&gt;</pre>
while (e) {a}	<pre>&lt;while&gt;   &lt;condition&gt; e &lt;/condition&gt;   a &lt;/while&gt;</pre>
seq a <sub>1</sub> ; ... ; a <sub>n</sub> qes	<pre>&lt;sequence&gt;   a<sub>1</sub> ... a<sub>n</sub> &lt;/sequence&gt;</pre>
flw a <sub>1</sub>   ...   a <sub>n</sub> wlf	<pre>&lt;flow&gt;   a<sub>1</sub> ... a<sub>n</sub> &lt;/flow&gt;</pre>
pck a <sub>1</sub> + ... + a <sub>n</sub> kcp	<pre>&lt;pick&gt;   a<sub>1</sub> ... a<sub>n</sub> &lt;/pick&gt;</pre>
[ a @ a <sub>f</sub> * a <sub>c</sub> ]	<pre>&lt;scope&gt;   &lt;faultHandlers&gt;     &lt;catchAll&gt;       &lt;sequence&gt;         &lt;compensate/&gt; a<sub>f</sub>       &lt;/sequence&gt;     &lt;/catchAll&gt;   &lt;/faultHandlers&gt;   &lt;compensationHandler&gt;     a<sub>c</sub>   &lt;/compensationHandler&gt;   a &lt;/scope&gt;</pre>

Table 5: Mapping of service definitions

<i>Blite</i>	WS-BPEL
[ a @ a <sub>f</sub> ]	<pre>&lt;process&gt;   &lt;faultHandlers&gt;     &lt;catchAll&gt;       &lt;sequence&gt;         &lt;compensate/&gt; a<sub>f</sub>       &lt;/sequence&gt;     &lt;/catchAll&gt;   &lt;/faultHandlers&gt;   a &lt;/process&gt;</pre>

or the development environment mentioned in Section 4.1. The WS-BPEL file included in the generated package, where irrelevant details have been omitted, is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="virtualcardProcess" ... >
  <import location="virtualcard.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" />
  </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 > 0</condition>
  <sequence>
    <receive partnerLink="cltPL"
      operation="o_getcash" variable="var1">
      <correlations>
        <correlation set="x_idCorr" initiate="no" />
      </correlations>
    </receive>
    <if>
      <condition>
        $var0.amount >= $var1.wdrAmount
      </condition>
      <sequence>
        <assign>
          <copy>
            <from>$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>
    </if>
    <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>
```

As expected, the resulting WS-BPEL code is more verbose and intricate than the *Blite* one. However, the performed translation turns out to be quite ‘clean’, in the sense that each *BliteC* activity has been translated into the corresponding WS-BPEL one without introducing ‘junk’ code. The WSDL document associated to the above WS-BPEL program is reported in the Appendix.

To deploy the file `virtualcardProcess.bpr`, 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`



activeBPEL® engine	
Home	Home
Engine	Engine
Configuration	Configuration
Storage	Storage
Version Detail	Version Detail
Extended Services	Extended Services
Identity Service	Identity Service
Deployment Status	Deployment Status
Deployment Log	Deployment Log
Deployed Processes	Deployed Processes
Deployed Services	Deployed Services
Partner Definitions	Partner Definitions
Resource Catalog	Resource Catalog
Process Status	Process Status
Active Processes	Active Processes
Alarm Queue	Alarm Queue
Receive Queue	Receive Queue
Process ID	Process ID
	<input type="text"/> <input type="button" value="Go"/>
Help	Help

Figure 5: List of deployed processes

should appear. Now, by selecting Deployed services, we can retrieve the URLs of the two WSDL files<sup>6</sup> corresponding to the 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
```

To test the service behaviour, we can use a tool for automatic generation of web service requests, as e.g. soapUI [13], and 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 request we obtain the same response, but the instance status changes to Completed.

## 5.2 On asynchronous communication

Frequently, it is assumed that a service request can be processed in a reasonable amount of time, justifying the requirement that the invoker waits for a response related to a

<sup>6</sup>In fact, when provided with a WSDL file, ActiveBPEL produces as many WSDL files as the different partner links.

synchronous request-response operation. In a business process setting, where communication costs are high or network latency is unpredictable, such assumption usually does not hold and the interactions are better modeled by asynchronous message exchanges. Therefore, an asynchronous messaging approach is considered good practice for web services and service orchestrations in particular.

Although WS-BPEL provides the means for implementing asynchronous communication, it requires programmers to directly deal with endpoint references, as shown by the example in Section 2. Instead, in *Blite* this relevant communication pattern can be easily and transparently implemented.

A *Blite* specification for receiving a request and asynchronously replying with the string Hello is the following:

```
s_asyncServer ::= [ seq
  rcv <server,client> request <x_id>;
  x := "Hello";
  inv <client> response <x_id,x>
  qes ];;
asyncServer ::= {s_asyncServer}{x_id};;
```

Since this service does not need to invoke other services, the declarative part is just as follows:

```
<?blm
  ADDRESSES {
    myns => "http://asyncComm";
    myaddress => "http://XXX:8080/active-bpel/services";
  }
  VARIABLES {
    <x_id> => gen:req,<id>,<xsd:int>;
    <x_id,x> => gen:resp,<id,msg>,<xsd:int,xsd:string>;
  }
?>
```

The corresponding invoker service is rendered in *Blite* as

```
s_asyncClient ::= [ seq
  rcv <p_init,y_clt> init <y_id>;
  inv <server,client> request <y_id>;
  rcv <client> response <y_id,y_msg>;
  y_resp := "Response: ".y_msg;
  inv <y_clt> init <y_id,y_resp>
  qes ];;
asyncClient ::= {s_asyncClient}{y_id};;
```

The process is initialized by invoking the synchronous request-response operation `init`. Then, the created instance asynchronously invokes the partner server and, once receives the reply message, appends it at the response and sends the obtained message to the initiator. Notably, in *Blite* synchronous and asynchronous interactions are rendered in a similar way (i.e. as pairs of activities `inv-rcv` and `rcv-inv`); they are distinguished only by the fact that synchronous interactions use the same operation for invoking and receiving.

The declarative part of the asynchronous invoker is:

```
<?blm
  ADDRESSES {
    myns => "http://asyncComm";
    myaddress => "http://XXX:8080/active-bpel/services";
  }
  IMPORTS { asS => "http://asyncComm/asyncServer.wsdl"; }
  VARIABLES {
    <y_id> => asS:req;
    <y_id,y_msg> => asS:resp;
    <y_id,y_resp> => gen:response,<id,resp>,<xsd:int,xsd:string>;
  }
  PARTNERLINKS {
    PARTNERLINK {
      TYPE => asS:serverPLT;
      MY_ROLE client => asS:clientPT;
      PARTNER_ROLE server => asS:serverPT;
    }
  }
?>
```

The types of the messages corresponding to the asynchronous invocation and the related response are imported by the WSDL document of the asynchronous server (identified by the prefix `ass`). Moreover, while the partner link to interact with the initiator partner `client` is automatically generated by *BliteC*, the partner link to interact with `server` must be explicitly declared (the types of the partner link and the involved ports are again imported from the server's WSDL document). Hence, the server must be compiled first.

We report here an excerpt of the WS-BPEL program corresponding to `asyncServer`:

```
<process name="asyncServerProcess" ... >
...
<partnerLinks>
  <partnerLink name="serverPL" partnerLinkType="mwl:serverPLT"
    myRole="server"
    partnerRole="client"
    initializePartnerRole="no" />
</partnerLinks>
<variables> ... </variables>
<correlationSets>
  <correlationSet name="x_idCorr" properties="mwl:x_idProp" />
</correlationSets>
<faultHandlers> ... </faultHandlers>
<sequence>
  <sequence>
    <receive partnerLink="serverPL"
      operation="request"
      variable="var0"
      createInstance="yes">
      <correlations>
        <correlation set="x_idCorr" initiate="yes" />
      </correlations>
    </receive>
    <assign> ... </assign>
  </sequence>
  <sequence>
    <receive partnerLink="serverPL"
      operation="setClientAddress"
      variable="var2">
      <correlations>
        <correlation set="x_idCorr" initiate="no" />
      </correlations>
    </receive>
    <assign>
      <copy>
        <from variable="var2" part="address" />
        <to partnerLink="serverPL" />
      </copy>
    </assign>
  </sequence>
  <assign> ... </assign>
  <invoke partnerLink="serverPL" operation="response" ... />
</sequence>
</process>
```

To enable asynchronous communication, the translation automatically puts an additional receive activity (highlighted by a gray background) in the generated WS-BPEL code. This activity will be used by clients to communicate their addresses, which are then assigned to the partner link used for the callback operation. Similarly, in a transparent way, *BliteC* equips the clients invoking this partner link with the symmetric invoke activity.

### 5.3 An auction service scenario

We show here an application of *BliteC* to a scenario built upon the auction service drawn from the WS-BPEL specification document and already introduced in Section 2.

The *auction service* is defined in *Blite* as:

```
s_auction ::=
[ seq
  flw
  rcv <auctionS,seller> submit
    <x_auctionId,x_creditCardNumberS,x_shippingCost>
  |
```

```
rcv <auctionB,buyer> submit
  <x_auctionId,x_creditCardNumberB,x_phoneNumber>
wlf;
x_amount := 1;
inv <register,auction> process <x_auctionId,x_amount>;
rcv <auction> regAnswer <x_auctionId,x_registrationId>;
x_resp := "Thank you!";
flw
  inv <seller> answer <x_auctionId,x_resp>
  |
  inv <buyer> answer <x_auctionId,x_resp>
wlf
qes ];;
```

```
auction ::= {s_auction}{x_auctionId};
```

```
<?blm
ADDRESSES {
  myns => "http://auction";
  myaddress => "http://localhost:8080/active-bpel/services";
}
IMPORTS { reg => "http://register/register.wsdl"; }
VARIABLES {
  <x_auctionId,x_creditCardNumberS,x_shippingCost>
  => gen:sellerReq,<id,ccNum,cost>,
  <xsd:int,xsd:string,xsd:int>;
  <x_auctionId,x_creditCardNumberB,x_phoneNumber>
  => gen:buyerReq,<id,ccNum,phone>,
  <xsd:int,xsd:string,xsd:string>;
  <x_auctionId,x_amount> => reg:regReq;
  <x_auctionId,x_registrationId> => reg:regResp;
  <x_auctionId,x_resp>
  => gen:resp,<id,msg>,<xsd:int,xsd:string>;
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => reg:registerPLT;
    MY_ROLE auction => reg:auctionPT;
    PARTNER_ROLE register => reg:registerPT;
  }
}
?>
```

Here, differently from the program in Section 2, the endpoint references of the registration, buyer and seller services are not explicitly handled by the programmer. Moreover, the only partner link declared is that used for interacting with the registration service.

The *registration service* is rendered in *Blite* as follows:

```
register ::=
{[ seq
  rcv <register,auction> process <x_id,x_amount>;
  x_registrationId := "someId";
  inv <auction> regAnswer <x_id,x_registrationId>
  qes ]}{x_id};

<?blm
ADDRESSES {
  myns => "http://register";
  myaddress => "http://localhost:8080/active-bpel/services";
}
VARIABLES {
  <x_id,x_amount>
  => gen:regReq,<id,amount>,<xsd:int,xsd:int>;
  <x_id,x_registrationId>
  => gen:regResp,<id,regId>,<xsd:int,xsd:string>;
}
?>
```

Its behaviour is very simple: the process gets instantiated by the auction service by invoking the operation `process`; then, the created instance replies with a registration identifier. For the sake of simplicity, we do not model here the generation of a unique identifier (which most likely could be provided by another service).

Finally, we report below the *seller service* (the *buyer service* is defined similarly):

```
v_seller ::=
[ seq
  rcv <initSeller,initiator> init <x_id,x_cc,x_shipCost>;
  inv <auction,seller> submit <x_id,x_cc,x_shipCost>;
  rcv <seller> answer <x_id,x_resp>;
  inv <initiator> init <x_id,x_resp>
```

```

    qes ];];
seller ::= {v_seller}{x_id};
<?blm ADDRESSES {
  myns => "http://seller";
  myaddress => "http://localhost:8080/active-bpel/services";
}
IMPORTS { as => "http://auction/auction.wsdl"; }
VARIABLES {
  <x_id,x_cc,x_shipCost> => as:sellerReq;
  <x_id, x_resp> => as:resp;
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => as:auctionSPLT;
    MY_ROLE seller => as:sellerPT;
    PARTNER_ROLE auction => as:auctionSPT;
  }
}
?>

```

The above specification is very similar to that of the asynchronous client described in Section 5.2.

Once the above programs have been compiled and deployed, regardless of the activation order of the buyer and seller, both services will receive the message `Thank you!` indicating the successful termination of the orchestration.

## 5.4 An auction service scenario with fault and compensation handling

We now extend the scenario introduced in Section 5.3 with fault and compensation handling. Basically, we allow the buyer and the seller services to cancel the auction, which causes its unregistration. Thus, the auction service becomes:

```

invReg ::=
  seq
  inv <register,auction> process <x_auctionId,x_amount>;
  rcv <auction> regAnswer <x_auctionId,x_registrationId>;
  flw
  inv <seller> answer <x_auctionId,x_resp>
  |
  inv <buyer> answer <x_auctionId,x_resp>
  wlf
  qes;;

compReg ::=
  seq
  x_motivation := "Auction cancelled";
  inv <registerComp> unregister <x_auctionId,x_motivation>
  qes;;

fh ::=
  seq
  x_resp := "The registration has been cancelled";
  flw
  inv <seller> answer <x_auctionId,x_resp>
  |
  inv <buyer> answer <x_auctionId,x_resp>
  wlf
  qes;;

s_auction ::=
  [ seq
  flw
  rcv <auctionS,seller> submit
  <x_auctionId,x_creditCardNumberS,x_shippingCost>
  |
  rcv <auctionB,buyer> submit
  <x_auctionId,x_creditCardNumberB,x_phoneNumber>
  wlf;
  x_amount := 1;
  x_resp := "Thank you!";
  [invReg @ empty * compReg];
  rcv <auctionCanc> cancel <x_auctionId>;
  throw
  qes
  @ fh];];

auction ::= {s_auction}{x_auctionId};

<?blm
ADDRESSES {
  myns => "http://auction";

```

```

  myaddress => "http://localhost:8080/active-bpel/services";
}
IMPORTS { reg => "http://register/register.wsdl"; }
VARIABLES {
  <x_auctionId,x_creditCardNumberS,x_shippingCost>
  => gen:sellerReq,<id,ccNum,cost>,
  <xsd:int,xsd:string,xsd:int>;
  <x_auctionId,x_creditCardNumberB,x_phoneNumber>
  => gen:buyerReq,<id,ccNum,phone>,
  <xsd:int,xsd:string,xsd:string>;
  <x_auctionId,x_amount> => reg:regReq;
  <x_auctionId,x_registrationId> => reg:regResp;
  <x_auctionId,x_motivation> => reg:unreg;
  <x_auctionId,x_resp>
  => gen:resp,<id,msg>,<xsd:int,xsd:string>;
  <x_auctionId> => gen:cancellation,<id>,<xsd:int>;
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => reg:registerPLT;
    MY_ROLE auction => reg:auctionPT;
    PARTNER_ROLE register => reg:registerPT;
  }
  PARTNERLINK {
    TYPE => reg:registerCompPLT;
    PARTNER_ROLE registerComp => reg:registerCompPT;
  }
}
?>

```

Once the auction service receives the two requests, it executes the scope activity `[invReg @ empty * compReg]`: the primary activity `invReg` interacts with the registration service and then replies to buyer and seller, while the activity `compReg` is the corresponding compensation handler that simply invokes the registration service to cancel the registration. After the scope completes, the auction service waits for a cancellation message by either the buyer or the seller. The reception of such a message generates a fault (through the activity `throw`). The fault is handled by the fault handler `fh` that automatically activates the compensation handler `compReg` (we refer to [24] for a complete account of the default compensation mechanism) and sends two messages to buyer and seller to notify that the registration has been cancelled.

## 5.5 A shipping scenario

We show here an application of *BliteC* to a scenario built upon the shipping service drawn from the official specification of WS-BPEL [29, Sect. 15.1]. The shipping service handles the shipment of orders. From the service point of view, orders are composed of a number of items. The service offers two types of shipments: shipments where the items are held and shipped together and shipments where the items are shipped piecemeal until the order is fulfilled. A skeleton description follows:

```

receive shipOrder
if (shipComplete) then send shipNotice
else
  itemsShipped := 0
  while (itemsShipped < itemsTotal) do
    itemsCount := opaque // non-deterministic assignment
                          // corresponding e.g. to interaction
                          // with a back-end system
    send shipNotice
    itemsShipped = itemsShipped + itemsCount

```

To generate an executable process, we have to replace the ‘opaque’ assignment with an invocation to the *back-end service*, that in *Blite* is rendered as follows:

```

s_backend ::=
  [ seq
  rcv <p_backend, x_client> o_num <x_id>;
  rcv <p_human> o_humanInteraction <x_id,x_num>;
  inv <x_client> o_num <x_num,x_id>
  qes ];];

backend_service ::= {s_backend}{x_id};

```

```

<?blm
ADDRESSES {
  myns => "http://example/backendService";
  myaddress => "http://XXX:8080/active-bpel/services";
}
VARIABLES {
  <x_id> => gen:id,<id>,<xsd:int>;
  <x_id,x_num> => gen:human,<id,num>,<xsd:int,xsd:int>;
  <x_num,x_id> => gen:number,<num,id>,<xsd:int,xsd:int>;
}
?>

```

The process gets instantiated by the shipping service by invoking the operation `o_num`; then, the created instance waits for an integer number (representing the quantity of available items) provided by a human actor along the operation `o_humanInteraction` and concludes by sending the number back to the shipping service. The order identifier, stored in `x_id`, is used as a correlation value.

The *shipping service* in *Blite* is defined as:

```

a_ship ::= seq
  x_shipped := 0;
  while (x_shipped < x_items) {
    seq
      inv <backend,cb_backend> o_num <x_id>;
      rcv <cb_backend> o_num <x_count,x_id>;
      if (x_count <= 0)
        { throw }
      else { seq
          inv <x_cust> o_notice <x_id,x_count>;
          x_shipped := x_shipped + x_count
        }
      qes
    }
  qes ;;

a_err ::= seq
  x_sorry:="Sorry, the required item is out of stock ";
  inv <x_cust> o_err <x_id,x_sorry>
  qes;;

s_ship ::=
[ seq
  rcv <p_ship, x_cust> o_req <x_id,x_c,x_items>;
  if (x_c > 0)
    { inv <x_cust> o_notice <x_id,x_items> }
  else { [a_ship @ a_err] }
  qes ];;

shipping_service ::= {s_ship}{x_id};

<?blm
ADDRESSES {
  myns => "http://example";
  myaddress => "http://XXX:8080/active-bpel/services";
}
IMPORTS { bck => "http://example/backendService/backend_service.wsdl"; }
VARIABLES {
  <x_id,x_c,x_items> => gen:shipOrder,
    <id, shipComplete, items>,
    <xsd:int, xsd:int, xsd:int>;
  <x_id,x_items> => gen:shippingNoticeMsg,
    <id, items>, <xsd:int,xsd:int>;
  <x_id,x_count> => gen:shippingNoticeMsg;
  <x_id,x_sorry> => gen:shippingErrorMsg,
    <id, errorMsg>,
    <xsd:int,xsd:string>;
  <x_id> => bck:id;
  <x_count,x_id> => bck:number;
  x_shipped => xsd:integer;
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => bck:clientPLT;
    PARTNER_ROLE backend => bck:p_backendPT;
  }
}
?>

```

Here, after the invocation of the back-end service, the returned number (stored in `x_count`) is checked: a number less than or equal to 0 means that the required item is out of stock and, hence, a fault is raised by means of the `throw` activity. The fault will be caught and handled by the fault handler `a_err`, that will send an error message to the client.

Finally, we report below a (dummy) *client service*:

```

s_shipClient ::=
[ seq
  rcv <p_init,y_clt> o_init <y_id,y_c, y_items>;
  y_resp := "ORDER: ". y_id . " BEGIN ";
  inv <ship_srv,cust> o_req <y_id,y_c, y_items>;
  y_shipped := 0;
  while(y_shipped < y_items) {
    pck
      rcv <cust> o_notice <y_id,y_count>;
      seq
        y_shipped := y_shipped + y_count;
        y_resp := y_resp."NOTICE: sent items=".y_count." ";
      qes
      +
      rcv <cust> o_err <y_id, y_sorry>;
      seq
        y_shipped := y_items;
        y_resp := y_resp."ERROR: ". y_sorry
      qes
      kcp };
  y_resp := y_resp . " END";
  inv <y_clt> o_init <y_id,y_resp>
  qes ];;

shipping_client ::= {s_shipClient}{y_id};

<?blm
ADDRESSES {
  myns => "http://example";
  myaddress => "http://XXX:8080/active-bpel/services";
}
IMPORTS { shs => "http://example/shipping_service.wsdl"; }
VARIABLES {
  <y_id, y_c, y_items> => shs:shipOrder;
  <y_id,y_sorry> => shs:shippingErrorMsg;
  <y_id,y_count> => shs:shippingNoticeMsg;
  y_shipped => xsd:integer;
  <y_id,y_resp> => gen:response,
    <id,resp>,
    <xsd:int,xsd:string>;
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => shs:custPLT;
    MY_ROLE cust => shs:x_custPT;
    PARTNER_ROLE ship_srv => shs:p_shipPT;
  }
}
?>

```

This client receives a shipping request and forwards it to the shipping service. Then, it waits all response messages, stores them in a string variable (i.e. `y_resp`) and sends back the string to the invoker.

Once the programs have been compiled and deployed, if we invoke the operation `o_init` provided by the client service by sending the request `<1234,0,7>` (i.e. we require 7 items shipped piecemeal with order identifier 1234) and manually specify that the shipment is divided in two packages of 3 and 4 items, we will get back the string:

```
ORDER: 1234 BEGIN NOTICE: sent items=3; NOTICE: sent items=4; END
```

## 5.6 Orchestrating non-Blite services

The previous examples show how *BliteC* can be used to generate complete orchestration scenarios where each service is obtained by a *Blite* specification. However, our tool can be also used to orchestrate *Blite* services together with non-*Blite* ones. To this aim, our specification language provides two simple constructs, `get` and `set`, that permit manipulating XML messages exchanged with (possibly) non-*Blite* services.

Consider a service that receives a US zip code and a preference for the temperature scale, i.e. either the character `c` for Celsius or `f` for Fahrenheit, contacts a first service for getting the current weather conditions and the temperature in degrees Fahrenheit of the city corresponding to the

zip code, possibly contacts a second service for converting degrees Fahrenheit into degrees Celsius, and finally sends the obtained weather and temperature information to the invoker. To implement this service we have orchestrated two web services freely provided by CDYNE [1]. The corresponding *Blite* specification is as follows:

```
zipWeatherClient ::=
{
  seq
  rcv <initWeatherClient,initiator> init <num,scale,reqWeath>;
  inv <weather, cb_weather> reqWeather <reqWeath>;
  rcv <cb_weather> reqWeather <respWeath>;
  x := get(respWeath, "/wth:GetCityWeatherByZIPResponse
                    /wth:GetCityWeatherByZIPResult
                    /wth:Description");
  x_city := get(respWeath, "/wth:GetCityWeatherByZIPResponse
                          /wth:GetCityWeatherByZIPResult
                          /wth:City");
  x_temF := get(respWeath, "/wth:GetCityWeatherByZIPResponse
                          /wth:GetCityWeatherByZIPResult
                          /wth:Temperature");
  x := "The current weather at " . x_city . " is: " . x .
      ", the temperature is: ";
  if (scale == "c")
  {seq
    reqConv := $litConv;
    set(reqConv, "/tem:FahrenheitToCelcius
               /tem:nFahrenheit") := x_temF;
    inv <converter, cb_converter> reqConversion <reqConv>;
    rcv <cb_converter> reqConversion <respConv>;
    x_temC := get(respConv, "/tem:FahrenheitToCelciusResponse
                          /tem:FahrenheitToCelciusResult");
    x := x . x_temC . " °C"
  }
  qes
  else {x := x . x_temF . " °F";
    inv <initiator> init <num,x>
  }
  qes }{num};
}

<?blm
ADDRESSES {
  myns => "http://zipWeather";
  myaddress => "http://localhost:8080/active-bpel/services";
}
IMPORTS {
  wth => "http://ws.cdyne.com/WeatherWS/";
  tem => "http://webservices.daehosting.com/temperature";
}
VARIABLES {
  <reqWeath> => wth:GetCityWeatherByZIPSoapIn;
  <respWeath> => wth:GetCityWeatherByZIPSoapOut;
  <num,scale,reqWeath>
    => gen:req, <init,scale,request>,
    <xsd:integer,xsd:string,(El)wth:GetCityWeatherByZIP>;
  <num,x>
    => gen:resp, <init,response>,<xsd:integer,xsd:string>;
  x_city => xsd:string;
  x_temF => xsd:decimal;
  <reqConv> => tem:FahrenheitToCelciusSoapRequest;
  <respConv> => tem:FahrenheitToCelciusSoapResponse;
  x_temC => xsd:string;
}
LITERALS {
  litConv => [[<tem:FahrenheitToCelcius
              xmlns:tem="http://webservices.daehosting.com
                    /temperature"
              <tem:nFahrenheit>0<tem:nFahrenheit>
              </tem:FahrenheitToCelcius>]];
}
PARTNERLINKS {
  PARTNERLINK {
    TYPE => gen:weatherPLT;
    PARTNER_ROLE weather => wth:WeatherSoap,
    (reqWeather =>GetCityWeatherByZIP);
  }
  PARTNERLINK {
    TYPE => gen:converterPLT;
    PARTNER_ROLE
    converter => tem:TemperatureConversionsSoapType,
    (reqConversion =>FahrenheitToCelcius);
  }
}
}>
```

The `get` construct is used here to extract information from the XML messages received from the two external web services, while the `set` construct is used to insert the received temperature expressed in degrees Fahrenheit into the XML

request message for the converter service, whose structure has been previously initialized by means of the literal `litConv`.

Once the program has been compiled and deployed, if we invoke the operation `init` by specifying the zip code 90210 and the scale Celsius, we will get back a string of the form:

```
The current weather at Beverly Hills is: Clear,
the temperature is: 14.444444 °C
```

## 6. CONCLUDING REMARKS

We have presented *BliteC*, a software tool for supporting a rapid and easy development of WS-BPEL applications. The tool aims at solving some well-known programming problems of WS-BPEL caused by its XML syntax, lack of a formal semantics, and non-standardization of the deployment procedure. Basically, *BliteC* takes as inputs programs written in *Blite*, a prototypical orchestration language inspired to WS-BPEL but with a simpler syntax and a well-defined operational semantics, and provides as output the corresponding deployable WS-BPEL programs.

The aim of facilitating the development of WS-BPEL applications is shared also by the several graphical editors that permit designing WS-BPEL processes, among which we mention the designers embedded in Oracle BPEL Process Manager [5], Intalio Designer [10], ActiveVOS Designer [7], and Eclipse BPEL designer [9]. Although their use is quite intuitive, developing large applications by using them can be awkward and annoying compared to the more classic textual approach. Indeed, graphical notations turn out to be suitable for beginner WS-BPEL programmers to represent simple business process workflows, but do not allow more expert programmers to exploit commonly used functionalities, such as e.g. copy/cut/paste, and are inappropriate for expressing some (textual) information, such as e.g. correlation sets. Moreover, graphical designers have a significant negative impact on performance during the programming phase (that is, indeed, the phase of the software development process on which we focus on), since they usually are plugins of heavy software development environments such as JDeveloper [3] and Eclipse [4]. Some other works with a similar aim are [27, 31, 14]. The first two present some tools that produce WS-BPEL processes starting, respectively, from UML- and Petri Nets-based representations of SOC applications. Due to the use of graphical representations, also these tools suffer from the problems previously mentioned. Instead, the third one proposes a mapping from a  $\pi$ -calculus based formalism into WS-BPEL. In all three approaches, only non-executable WS-BPEL processes are generated, i.e. the generated code should be thought of as a template code where, besides binding and deployment details, programmers have also to define things such as partner links, variables, port types, correlations sets, etc. by editing the generated files. Another related work is [28], which proposes a different approach to develop SOC applications that still relies on a formal language. However, input programs are directly executed in a purposely developed engine, rather than being translated into and deployed as WS-BPEL processes.

Currently, the WS-BPEL packages generated by *BliteC* are intended to be deployed on ActiveBPEL. This is just to demonstrate feasibility of our approach. In fact, *BliteC* has

been designed so that the generation of deployment descriptors for different engines can be easily integrated, and we plan to enable it to produce packages also for other freely available engines, such as Oracle BPEL Process Manager, Apache ODE [8] and Beepell [22]. Of course, to preserve the semantics of the original *Blite* programs, one has to study the inner implementation of every supported engine and to define a customized translation. Since no engine has a formal description of its behaviour, this study has to be carried out by means of experimental tests and, most of all, no formal proof of semantics preservation can be done. It is also worth noticing that the semantics of *Blite*, which is in fact quite close to that of ActiveBPEL, could be rather tough to render by some WS-BPEL engines, whose semantics may significantly differ on low-level implementation details (e.g. message queue handling) or may more strictly (or inappropriately) enforce some WS-BPEL constraints. For instance, it may happen that a process instance should receive a message from a partner according to the *Blite* semantics, while instead the message cannot be effectively accepted according to the semantics of the considered engine, due to e.g. some peculiar correlation constraint. In such a case, *BliteC* should identify the potential conflicting receives in the generated WS-BPEL program and, e.g., replace them by a single receive enabling some proper coordination activities.

We also plan to enrich the *BliteC* development environment presented in Section 4.1 with further functionalities, such as deployment/undeployment facilities over remote servers, and debugging tools, such as automatic generation of web interfaces for invoking the created services and log recording based on an embedded dedicated web service. This latter tool could require to extend the syntax of *Blite* accepted by *BliteC* with a construct for printing strings into the log that would be translated into a WS-BPEL one-way interaction.

For what concern the language *Blite*, we also intend to investigate its extension to cover some WS-BPEL constructs that at the time being have been left out, such as timed activities, event and termination handlers, and more sophisticated forms of fault and compensation handling involving named faults and compensation of specified scopes. We do not envisage any major issue in translating such constructs in WS-BPEL code, while their addition to *Blite* would require to significantly revise the formal definition of the operational semantics of the language.

Finally, we intend to develop formal analysis techniques, e.g. based on model checking (as in [14, 15]), for *Blite* specifications. This way, we would be able to specify in *Blite* an orchestration scenario, validate its behaviour by using formal tools, and deploy it as a set of WS-BPEL programs.

## 7. ACKNOWLEDGMENTS

We thank Alessandro Lapadula for his fundamental contribution to the definition of the language *Blite*. We also thank the anonymous reviewers of SAC'10 and Applied Computing Review for their helpful comments, and people attending the SOAP track at SAC'10 for the fruitful discussions following the presentation of our work. Finally, we thank the chairs of the SOAP track for their invitation to submit to the Applied Computing Review and their support.

This work has been sponsored by the EU project SENSORIA, IST-2005-016004.

## 8. REFERENCES

- [1] CDYNE Corporation. <http://www.cdyne.com/>.
- [2] jEdit Programmer's Text Editor 4.3. <http://www.jedit.org/>.
- [3] Oracle JDeveloper. <http://www.oracle.com/technology/products/jdev>.
- [4] The Eclipse project. <http://www.eclipse.org>.
- [5] Oracle BPEL Process Manager 10.1.3, December 2007. <http://www.oracle.com/technology/bpel>.
- [6] ActiveBPEL 5.0.2, October 2009. <http://sourceforge.net/projects/activebpel502/>.
- [7] ActiveVOS Designer 5.0.2, June 2009. <http://www.activevos.com/>.
- [8] Apache ODE 1.3.3, August 2009. <http://ode.apache.org>.
- [9] Eclipse BPEL project 0.4.0, May 2009. <http://www.eclipse.org/bpel>.
- [10] Intalio|Designer Community Edition 6.0.1, August 2009. <http://www.intalio.com/products/bpm/community-edition/designer>.
- [11] JavaCC 4.2, April 2009. <https://javacc.dev.java.net>.
- [12] JDOM 1.1, April 2009. <http://www.jdom.org>.
- [13] soapUI 3.5, March 2010. <http://www.soapui.org>.
- [14] F. Abouzaid and J. Mullins. A Calculus for Generation, Verification and Refinement of BPEL Specifications. In *WWV*, volume 200 of *ENTCS*, pages 43–65. Elsevier, 2008.
- [15] F. Abouzaid and J. Mullins. Model-checking Web Services Orchestration using BP-calculus. In *FOCLASA*, volume 255 of *ENTCS*, pages 3–21. Elsevier, 2009.
- [16] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, 2001.
- [17] N. Dragoni and M. Mazzara. A Formal Semantics for the WS-BPEL Recovery Framework: The pi-Calculus Way. In *WS-FM*, volume 6194 of *LNCS*, pages 92–109. Springer, 2010.
- [18] G. Erich, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [19] D. Fahland and W. Reisig. ASM-based Semantics for BPEL: The Negative Control Flow. In *Abstract State Machines*, pages 131–152, 2005.
- [20] A. Ferrara. Web services: a process algebra approach. In *ICSOC*, pages 242–251. ACM, 2004.
- [21] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - Core. Technical report, W3C, May 2006. W3C Recommendation.
- [22] T. Hallwyl, F. Henglein, and T. Hildebrandt. A standard-driven implementation of WS-BPEL 2.0. In *SAC*, pages 2472–2476. ACM, 2010.
- [23] S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In *Business Process Management*, volume 3649 of *LNCS*, pages 220–235. Springer, 2005.
- [24] A. Lapadula, R. Pugliese, and F. Tiezzi. A formal account of WS-BPEL. In *COORDINATION*, volume

- 5052 of *LNCS*, pages 199–215. Springer, 2008.
- [25] N. Lohmann. A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In *WS-FM*, volume 4937 of *LNCS*, pages 77–91. Springer, 2008.
- [26] R. Lucchi and M. Mazzara. A pi-calculus based semantics for WS-BPEL. *J. Log. Algebr. Program.*, 70(1):96–118, 2007.
- [27] P. Mayer, A. Schroeder, and N. Koch. Mdd4soa: Model-driven service orchestration. In *EDOC*, pages 203–212. IEEE, 2008.
- [28] F. Montesi, C. Guidi, and G. Zavattaro. Composing Services with JOLIE. In *ECOWS*, pages 13–22. IEEE, 2007.
- [29] OASIS WSBPEL TC. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, April 2007.
- [30] C. Ouyang, E. Verbeek, W. van der Aalst, S. Breutel, M. Dumas, and A. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Sci. Comput. Program.*, 67(2-3):162–198, 2007.
- [31] W. M. P. van der Aalst and K. B. Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Information & Software Technology*, 50(3):131–159, 2008.
- [32] M. Weidlich, G. Decker, and M. Weske. Efficient Analysis of BPEL 2.0 Processes Using  $\pi$ -calculus. In *APSCC*, pages 266–274. IEEE, 2007.

## APPENDIX

### WS-BPEL and WSDL files generated by BliteC

We report here the WS-BPEL and WSDL files generated by *BliteC* for the examples shown in Section 5.

#### Virtual credit card service (WS-BPEL file)

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="virtualcardProcess"
  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"
  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" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <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 > 0 </condition>
    <sequence>
      <receive partnerLink="cltPL" operation="o_getcash"
        variable="var1">
        <correlations>
          <correlation set="x_idCorr" initiate="no" />
        </correlations>
      </receive>
      <if>
        <condition>
          $var0.amount >= $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>
      </if>
    </sequence>
  </while>
</process>
```

```

        </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 credit card service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<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 style="document"
        soapAction="http://example/virtualcard.wsdl/o_newcard" />
      <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 style="document"
        soapAction="http://example/virtualcard.wsdl/o_getcash" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>

```

```

    </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>

```

### Asynchronous server (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="asyncServerProcess"
  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://asyncComm/asyncServer.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://asyncComm/asyncServer.bpel">
  <import location="asyncServer.wsdl"
    namespace="http://asyncComm/asyncServer.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="serverPL"
      partnerLinkType="mwl:serverPLT"
      myRole="server"
      partnerRole="client"
      initializePartnerRole="no" />
  </partnerLinks>
  <variables>
    <variable name="var0" messageType="mwl:req" />
    <variable name="var2"
      messageType="mwl:clientAddressSERVER" />
    <variable name="var1" messageType="mwl:resp" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr" properties="mwl:x_idProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>
    <sequence>
      <receive partnerLink="serverPL" operation="request"
        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="corrID" />
  </copy>
</assign>
</sequence>
</sequence>
<receive partnerLink="serverPL"
  operation="setClientAddress"
  variable="var2">
  <correlations>
    <correlation set="x_idCorr" initiate="no" />
  </correlations>
</receive>
<assign>
  <copy>
    <from variable="var2" part="address" />
    <to partnerLink="serverPL" />
  </copy>
</assign>
</sequence>
<assign>
  <copy>
    <from>'Hello'</from>
    <to variable="var1" part="msg" />
  </copy>
</assign>
<invoke operation="response" inputVariable="var1"
  partnerLink="serverPL">
  <correlations>
    <correlation set="x_idCorr" pattern="request"
      initiate="no" />
  </correlations>
</invoke>
</sequence>
</process>

```

### Asynchronous server (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:tns="http://asyncComm/asyncServer.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://asyncComm/asyncServer.wsdl">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://asyncComm/asyncServer.wsdl">
      <xsd:import
        namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        schemaLocation="../../schema/ws-addressing.xsd" />
      <xsd:element name="xEL" type="xsd:string" />
      <xsd:element name="x_idEL" type="xsd:int" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="resp">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="msg" element="tns:xEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressSERVER">
    <wsdl:part name="corrID" element="tns:x_idEL" />
    <wsdl:part name="address" element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:message name="req">
    <wsdl:part name="id" element="tns:x_idEL" />
  </wsdl:message>
  <wsdl:portType name="clientPT">
    <wsdl:operation name="request">
      <wsdl:input message="tns:resp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="serverPT">
    <wsdl:operation name="response">
      <wsdl:input message="tns:req" />
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <wsdl:input message="tns:clientAddressSERVER" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="serverBinding" type="tns:serverPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />

```

```

    <wsdl:operation name="request">
      <soap:operation
        soapAction="http://asyncComm/asyncServer.wsdl/request"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <soap:operation
        soapAction="http://asyncComm/asyncServer.wsdl
          /setClientAddress"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="serverService">
    <wsdl:port name="serverPort" binding="tns:serverBinding">
      <soap:address
        location="http://localhost:8080/active-bpel/services
          /serverService" />
    </wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="serverPLT">
    <plnk:role name="server" portType="tns:serverPT" />
    <plnk:role name="client" portType="tns:clientPT" />
  </plnk:partnerLinkType>
  <prop:property name="x_idProp" type="xsd:int" />
  <prop:propertyAlias propertyName="tns:x_idProp"
    messageType="tns:clientAddressSERVER"
    part="corrID" />
  <prop:propertyAlias propertyName="tns:x_idProp"
    messageType="tns:resp"
    part="id" />
  <prop:propertyAlias propertyName="tns:x_idProp"
    messageType="tns:req"
    part="id" />
</wsdl:definitions>

```

### Asynchronous client (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="asyncClientProcess"
  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://asyncComm/asyncClient.wsdl"
  xmlns:asS="http://asyncComm/asyncServer.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://asyncComm/asyncClient.bpel">
  <import location="asyncServer.wsdl"
    namespace="http://asyncComm/asyncServer.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <import location="asyncClient.wsdl"
    namespace="http://asyncComm/asyncClient.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="server"
      partnerLinkType="asS:serverPLT"
      myRole="client"
      partnerRole="server" />
    <partnerLink name="cltPL"
      partnerLinkType="mwl:cltPLT"
      myRole="p_init" />
  </partnerLinks>
  <variables>
    <variable name="var1" messageType="asS:resp" />
    <variable name="var3"
      messageType="asS:clientAddressSERVER" />
    <variable name="var0" messageType="asS:req" />
    <variable name="var2" messageType="mwl:response" />
  </variables>
  <correlationSets>
    <correlationSet name="y_idCorr" properties="mwl:y_idProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>

```

```

<sequence>
  <receive partnerLink="cltPL" operation="init"
    variable="var0" createInstance="yes">
    <correlations>
      <correlation set="y_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>
    <copy>
      <from variable="var0" part="id" />
      <to variable="var3" part="corrID" />
    </copy>
  </assign>
</sequence>
<invoke operation="request" inputVariable="var0"
  partnerLink="server">
  <correlations>
    <correlation set="y_idCorr" pattern="request"
      initiate="no" />
  </correlations>
</invoke>
<sequence>
  <assign>
    <copy>
      <from>
        <literal>
          <wsa:EndpointReference
            xmlns:wsa="http://schemas.xmlsoap.org/ws
              /2003/03/addressing"
            xmlns:dyn="http://asyncComm/asyncServer.wsdl">
            <wsa:Address>http://localhost:8080/active-bpel
              /services/clientService</wsa:Address>
            <wsa:PortType>dyn:clientPT</wsa:PortType>
            <wsa:ServiceName PortName="clientServicePort">
              dyn:clientService
            </wsa:ServiceName>
          </wsa:EndpointReference>
        </literal>
      </from>
      <to variable="var3" part="address" />
    </copy>
  </assign>
  <invoke operation="setClientAddress" inputVariable="var3"
    partnerLink="server">
    <correlations>
      <correlation set="y_idCorr" pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
</sequence>
<receive partnerLink="server" operation="response"
  variable="var1">
  <correlations>
    <correlation set="y_idCorr" initiate="no" />
  </correlations>
</receive>
<assign>
  <copy>
    <from>concat('Response: ',string($var1.msg))</from>
    <to variable="var2" part="resp" />
  </copy>
  <assign>
    <reply operation="init" partnerLink="cltPL" variable="var2">
    <correlations>
      <correlation set="y_idCorr" initiate="no" />
    </correlations>
    </reply>
  </sequence>
</process>
  xmlns:prop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
  targetNamespace="http://asyncComm/asyncClient.wsdl">
  <wsdl:import namespace="http://asyncComm/asyncServer.wsdl"
    location="asyncServer.wsdl" />
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://asyncComm/asyncClient.wsdl">
      <xsd:element name="y_respEL" type="xsd:string" />
      <xsd:element name="y_idEL" type="xsd:int" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="response">
    <wsdl:part name="id" element="tns:y_idEL" />
    <wsdl:part name="resp" element="tns:y_respEL" />
  </wsdl:message>
  <wsdl:portType name="p_initPT">
    <wsdl:operation name="init">
      <wsdl:input message="asS:req" />
      <wsdl:output message="tns:response" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="p_initBinding" type="tns:p_initPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="init">
      <soap:operation
        soapAction="http://asyncComm/asyncClient.wsdl/init"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="clientBinding" type="asS:clientPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="response">
      <soap:operation
        soapAction="http://asyncComm/asyncServer.wsdl/response"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="p_initService">
    <wsdl:port name="p_initPort" binding="tns:p_initBinding">
      <soap:address
        location="http://localhost:8080/active-bpel
          /services/p_initService" />
    </wsdl:port>
  </wsdl:service>
  <wsdl:service name="clientService">
    <wsdl:port name="clientPort" binding="tns:clientBinding">
      <soap:address
        location="http://localhost:8080/active-bpel
          /services/clientService" />
    </wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="cltPLT">
    <plnk:role name="p_init" portType="tns:p_initPT" />
  </plnk:partnerLinkType>
  <prop:property name="y_idProp" type="xsd:int" />
  <prop:propertyAlias propertyName="tns:y_idProp"
    messageType="asS:clientAddressSERVER"
    part="corrID" />
  <prop:propertyAlias propertyName="tns:y_idProp"
    messageType="tns:response"
    part="id" />
  <prop:propertyAlias propertyName="tns:y_idProp"
    messageType="asS:resp"
    part="id" />
  <prop:propertyAlias propertyName="tns:y_idProp"
    messageType="asS:req"
    part="id" />
</wsdl:definitions>

```

### Asynchronous client (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:asS="http://asyncComm/asyncServer.wsdl"
  xmlns:tns="http://asyncComm/asyncClient.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"

```

### Auction service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="auctionProcess"
  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://auction/auction.wsdl"
xmlns:reg="http://register/register.wsdl"
suppressJoinFailure="yes"
targetNamespace="http://auction/auction.bpel">
<import location="register.wsdl"
  namespace="http://register/register.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/" />
<import location="auction.wsdl"
  namespace="http://auction/auction.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/" />
<partnerLinks>
  <partnerLink name="auctionBPL"
    partnerLinkType="mwl:auctionBPL"
    myRole="auctionB"
    partnerRole="buyer"
    initializePartnerRole="no" />
  <partnerLink name="register"
    partnerLinkType="reg:registerPLT"
    myRole="auction"
    partnerRole="register" />
  <partnerLink name="auctionSPL"
    partnerLinkType="mwl:auctionSPL"
    myRole="auctionS"
    partnerRole="seller"
    initializePartnerRole="no" />
</partnerLinks>
<variables>
  <variable name="var5"
    messageType="mwl:clientAddressAUCTIONS" />
  <variable name="var7"
    messageType="reg:clientAddressREGISTER" />
  <variable name="var0" messageType="mwl:sellerReq" />
  <variable name="var4" messageType="mwl:resp" />
  <variable name="var2" messageType="reg:regReq" />
  <variable name="var6"
    messageType="mwl:clientAddressAUCTIONB" />
  <variable name="var1" messageType="mwl:buyerReq" />
  <variable name="var3" messageType="reg:regResp" />
</variables>
<correlationSets>
  <correlationSet name="x_auctionIdCorr"
    properties="mwl:x_auctionIdProp" />
</correlationSets>
<faultHandlers>
  <catchAll>
    <sequence>
      <compensate />
      <empty />
    </sequence>
  </catchAll>
</faultHandlers>
<sequence>
  <flow>
    <sequence>
      <sequence>
        <receive partnerLink="auctionSPL"
          operation="submit"
          variable="var0"
          createInstance="yes">
          <correlations>
            <correlation set="x_auctionIdCorr"
              initiate="join" />
          </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>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var3" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var4" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var5" part="corrID" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var6" part="corrID" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var7" part="corrID" />
          </copy>
        </assign>
      </sequence>
      <receive partnerLink="auctionBPL"
        operation="submit"
        variable="var1"
        createInstance="yes">
        <correlations>
          <correlation set="x_auctionIdCorr"
            initiate="join" />
        </correlations>
      </receive>
      <sequence>
        <receive partnerLink="auctionSPL"
          operation="setClientAddress"
          variable="var5">
          <correlations>
            <correlation set="x_auctionIdCorr"
              initiate="no" />
          </correlations>
        </receive>
        <assign>
          <copy>
            <from variable="var5" part="address" />
            <to partnerLink="auctionSPL" />
          </copy>
        </assign>
      </sequence>
      <sequence>
        <receive partnerLink="auctionBPL"
          operation="submit"
          variable="var1"
          createInstance="yes">
          <correlations>
            <correlation set="x_auctionIdCorr"
              initiate="join" />
          </correlations>
        </receive>
        <sequence>
          <receive partnerLink="auctionBPL"
            operation="setClientAddress"
            variable="var6">
            <correlations>
              <correlation set="x_auctionIdCorr"
                initiate="no" />
            </correlations>
          </receive>
          <assign>
            <copy>
              <from variable="var6" part="address" />
              <to partnerLink="auctionBPL" />
            </copy>
          </assign>
        </sequence>
      </sequence>
      <flow>
        <assign>
          <copy>
            <from>1</from>
            <to variable="var2" part="amount" />
          </copy>
        </assign>
        <invoke operation="process"
          inputVariable="var2"
          partnerLink="register">
          <correlations>
            <correlation set="x_auctionIdCorr"
              pattern="request" initiate="no" />
          </correlations>
        </invoke>
        <sequence>
          <assign>
            <copy>
              <from>
                <literal>
                  <wsa:EndpointReference
                    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
                    xmlns:dyn="http://register/register.wsdl">
                    <wsa:Address>http://localhost:8080/active-bpel/services/auctionService</wsa:Address>
                    <wsa:PortType>dyn:auctionPT</wsa:PortType>
                    <wsa:ServiceName
                      PortName="auctionServicePort">
                      dyn:auctionService
                    </wsa:ServiceName>
                    </wsa:EndpointReference>
                  </literal>
                </from>
              <to variable="var7" part="address" />
            </copy>
          </assign>
          <invoke operation="setClientAddress"
            inputVariable="var7"
            partnerLink="register">
          <correlations>
            <correlation set="x_auctionIdCorr"
              pattern="request" initiate="no" />
          </correlations>
        </invoke>
      </flow>
    </sequence>
  </flow>
</sequence>

```

```

</sequence>
<receive partnerLink="register"
  operation="regAnswer"
  variable="var3">
  <correlations>
    <correlation set="x_auctionIdCorr"
      initiate="no" />
  </correlations>
</receive>
<assign>
  <copy>
    <from>'Thank you! '</from>
    <to variable="var4" part="msg" />
  </copy>
</assign>
<flow>
  <invoke operation="answer"
    inputVariable="var4"
    partnerLink="auctionSPL">
    <correlations>
      <correlation set="x_auctionIdCorr"
        pattern="request" initiate="no" />
    </correlations>
  </invoke>
  <invoke operation="answer"
    inputVariable="var4"
    partnerLink="auctionBPL">
    <correlations>
      <correlation set="x_auctionIdCorr"
        pattern="request" initiate="no" />
    </correlations>
  </invoke>
</flow>
</sequence>
</process>

```

### Auction service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003
    /03/addressing"
  xmlns:reg="http://register/register.wsdl"
  xmlns:tns="http://auction/auction.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://auction/auction.wsdl">
  <wsdl:import namespace="http://register/register.wsdl"
    location="register.wsdl" />
  <wsdl:types>
    <xsd:schema targetNamespace="http://auction/auction.wsdl">
      <xsd:import
        namespace="http://schemas.xmlsoap.org/ws
          /2003/03/addressing"
        schemaLocation="../schema/ws-addressing.xsd" />
      <xsd:element name="x_phoneNumberEL"
        type="xsd:string" />
      <xsd:element name="x_creditCardNumberBEL"
        type="xsd:string" />
      <xsd:element name="x_auctionIdEL"
        type="xsd:int" />
      <xsd:element name="x_respEL"
        type="xsd:string" />
      <xsd:element name="x_shippingCostEL"
        type="xsd:int" />
      <xsd:element name="x_creditCardNumberSEL"
        type="xsd:string" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="buyerReq">
    <wsdl:part name="id" element="tns:x_auctionIdEL" />
    <wsdl:part name="ccNum" element="tns:x_creditCardNumberBEL" />
    <wsdl:part name="phone" element="tns:x_phoneNumberEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressAUCTIONB">
    <wsdl:part name="corrID" element="tns:x_auctionIdEL" />
    <wsdl:part name="address" element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:message name="resp">
    <wsdl:part name="id" element="tns:x_auctionIdEL" />
    <wsdl:part name="msg" element="tns:x_respEL" />
  </wsdl:message>
  <wsdl:message name="sellerReq">

```

```

    <wsdl:part name="id" element="tns:x_auctionIdEL" />
    <wsdl:part name="ccNum" element="tns:x_creditCardNumberSEL" />
    <wsdl:part name="cost" element="tns:x_shippingCostEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressAUCTIONS">
    <wsdl:part name="corrID" element="tns:x_auctionIdEL" />
    <wsdl:part name="address" element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:portType name="sellerPT">
    <wsdl:operation name="answer">
      <wsdl:input message="tns:resp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="auctionSPT">
    <wsdl:operation name="submit">
      <wsdl:input message="tns:sellerReq" />
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <wsdl:input message="tns:clientAddressAUCTIONS" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="buyerPT">
    <wsdl:operation name="answer">
      <wsdl:input message="tns:resp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="auctionBPT">
    <wsdl:operation name="submit">
      <wsdl:input message="tns:buyerReq" />
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <wsdl:input message="tns:clientAddressAUCTIONB" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="auctionSBinding" type="tns:auctionSPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="submit">
      <soap:operation
        soapAction="http://auction/auction.wsdl/submit"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <soap:operation
        soapAction="http://auction/auction.wsdl
          /setClientAddress"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="auctionBinding" type="reg:auctionPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="regAnswer">
      <soap:operation
        soapAction="http://register/register.wsdl
          /regAnswer"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="auctionBBinding" type="tns:auctionBPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="submit">
      <soap:operation
        soapAction="http://auction/auction.wsdl/submit"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <soap:operation
        soapAction="http://auction/auction.wsdl
          /setClientAddress"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="auctionSService">
    <wsdl:port name="auctionSport"
      binding="tns:auctionSBinding">
      <soap:address

```

```

        location="http://localhost:8080/active-bpel
        /services/auctionService" />
</wsdl:port>
</wsdl:service>
<wsdl:service name="auctionService">
  <wsdl:port name="auctionPort"
    binding="tns:auctionBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
      /services/auctionService" />
    </wsdl:port>
  </wsdl:service>
<wsdl:service name="auctionBService">
  <wsdl:port name="auctionBPort"
    binding="tns:auctionBBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
      /services/auctionBService" />
    </wsdl:port>
  </wsdl:service>
<plnk:partnerLinkType name="auctionSPLT">
  <plnk:role name="auctionS" portType="tns:auctionSPT" />
  <plnk:role name="seller" portType="tns:sellerPT" />
</plnk:partnerLinkType>
<plnk:partnerLinkType name="auctionBPLT">
  <plnk:role name="auctionB" portType="tns:auctionBPT" />
  <plnk:role name="buyer" portType="tns:buyerPT" />
</plnk:partnerLinkType>
<prop:property name="x_auctionIdProp" type="xsd:int" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="reg:clientAddressREGISTER"
  part="corrID" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="tns:clientAddressAUCTIONB"
  part="corrID" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="tns:clientAddressAUCTIONS"
  part="corrID" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="tns:resp"
  part="id" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="reg:regResp"
  part="id" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="reg:regReq"
  part="id" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="tns:buyerReq"
  part="id" />
<prop:propertyAlias propertyName="tns:x_auctionIdProp"
  messageType="tns:sellerReq"
  part="id" />
</wsdl:definitions>

```

### Auction registration service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="registerProcess"
  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://register/register.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://register/register.bpel">
  <import location="register.wsdl"
    namespace="http://register/register.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="registerPL"
      partnerLinkType="mwl:registerPLT"
      myRole="register"
      partnerRole="auction"
      initializePartnerRole="no" />
  </partnerLinks>
  <variables>
    <variable name="var1" messageType="mwl:regResp" />
    <variable name="var0" messageType="mwl:regReq" />
    <variable name="var2"
      messageType="mwl:clientAddressREGISTER" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr"
      properties="mwl:x_idProp" />
  </correlationSets>
</faultHandlers>

```

```

<catchAll>
  <sequence>
    <compensate />
    <empty />
  </sequence>
</catchAll>
</faultHandlers>
<sequence>
  <sequence>
    <receive partnerLink="registerPL"
      operation="process"
      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="corrID" />
      </copy>
    </assign>
  </sequence>
  <sequence>
    <receive partnerLink="registerPL"
      operation="setClientAddress"
      variable="var2">
      <correlations>
        <correlation set="x_idCorr"
          initiate="no" />
      </correlations>
    </receive>
    <assign>
      <copy>
        <from variable="var2" part="address" />
        <to partnerLink="registerPL" />
      </copy>
    </assign>
  </sequence>
  <assign>
    <copy>
      <from>'someId'</from>
      <to variable="var1" part="regId" />
    </copy>
  </assign>
  <invoke operation="regAnswer"
    inputVariable="var1"
    partnerLink="registerPL">
    <correlations>
      <correlation set="x_idCorr"
        pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
</sequence>
</process>

```

### Auction registration service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003
  /03/addressing"
  xmlns:tns="http://register/register.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://register/register.wsdl">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://register/register.wsdl">
      <xsd:import namespace="http://schemas.xmlsoap.org/ws
        /2003/03/addressing"
        schemaLocation=".. /schema/ws-addressing.xsd" />
      <xsd:element name="x_idEL" type="xsd:int" />
      <xsd:element name="x_amountEL" type="xsd:int" />
      <xsd:element name="x_registrationIdEL"
        type="xsd:string" />
    </xsd:schema>
  </wsdl:types>

```

```

</xsd:schema>
</wsdl:types>
<wsdl:message name="clientAddressREGISTER">
  <wsdl:part name="corrID" element="tns:x_idEL" />
  <wsdl:part name="address"
    element="wsa:EndpointReference" />
</wsdl:message>
<wsdl:message name="regReq">
  <wsdl:part name="id" element="tns:x_idEL" />
  <wsdl:part name="amount"
    element="tns:x_amountEL" />
</wsdl:message>
<wsdl:message name="regResp">
  <wsdl:part name="id" element="tns:x_idEL" />
  <wsdl:part name="regId"
    element="tns:x_registrationIdEL" />
</wsdl:message>
<wsdl:portType name="auctionPT">
  <wsdl:operation name="regAnswer">
    <wsdl:input message="tns:regResp" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="registerPT">
  <wsdl:operation name="process">
    <wsdl:input message="tns:regReq" />
  </wsdl:operation>
  <wsdl:operation name="setClientAddress">
    <wsdl:input message="tns:clientAddressREGISTER" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="registerBinding"
  type="tns:registerPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="process">
    <soap:operation
      soapAction="http://register/register.wsdl
        /process"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="setClientAddress">
    <soap:operation
      soapAction="http://register/register.wsdl
        /setClientAddress"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="registerService">
  <wsdl:port name="registerPort"
    binding="tns:registerBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/registerService" />
  </wsdl:port>
</wsdl:service>
<plnk:partnerLinkType name="registerPLT">
  <plnk:role name="register"
    portType="tns:registerPT" />
  <plnk:role name="auction"
    portType="tns:auctionPT" />
</plnk:partnerLinkType>
<prop:property name="x_idProp" type="xsd:int" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:clientAddressREGISTER"
  part="corrID" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:regResp" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:regReq" part="id" />
</wsdl:definitions>

```

### Buyer service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="buyerProcess"
  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://buyer/buyer.wsdl"
  xmlns:as="http://auction/auction.wsdl"

```

```

  suppressJoinFailure="yes"
  targetNamespace="http://buyer/buyer.bpel">
  <import location="auction.wsdl"
    namespace="http://auction/auction.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <import location="buyer.wsdl"
    namespace="http://buyer/buyer.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="initBuyerPL"
      partnerLinkType="mwl:initBuyerPLT"
      myRole="initBuyer" />
    <partnerLink name="auction"
      partnerLinkType="as:auctionBPLT"
      myRole="buyer"
      partnerRole="auctionB" />
  </partnerLinks>
  <variables>
    <variable name="var1" messageType="as:resp" />
    <variable name="var2"
      messageType="as:clientAddressAUCTIONB" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr"
      properties="mwl:x_idProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>
    <receive partnerLink="initBuyerPL" operation="init"
      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="corrID" />
      </copy>
    </assign>
  </sequence>
  <invoke operation="submit" inputVariable="var0"
    partnerLink="auction">
    <correlations>
      <correlation set="x_idCorr"
        pattern="request" initiate="no" />
    </correlations>
  </invoke>
  <sequence>
    <assign>
      <copy>
        <from>
          <literal>
            <wsa:EndpointReference
              xmlns:wsa="http://schemas.xmlsoap.org/ws
                /2003/03/addressing"
              xmlns:dyn="http://auction/auction.wsdl">
              <wsa:Address>http://localhost:8080
                /active-bpel/services/buyerService
            </wsa:Address>
            <wsa:PortType>dyn:buyerPT</wsa:PortType>
            <wsa:ServiceName
              PortName="buyerServicePort">
              dyn:buyerService
            </wsa:ServiceName>
          </wsa:EndpointReference>
        </literal>
      </from>
      <to variable="var2" part="address" />
    </copy>
  </assign>
  <invoke operation="setClientAddress"
    inputVariable="var2"
    partnerLink="auction">
    <correlations>
      <correlation set="x_idCorr"
        pattern="request" initiate="no" />
    </correlations>
  </invoke>
</sequence>

```

```

<receive partnerLink="auction"
  operation="answer" variable="var1">
  <correlations>
    <correlation set="x_idCorr" initiate="no" />
  </correlations>
</receive>
<reply operation="init" partnerLink="initBuyerPL"
  variable="var1">
  <correlations>
    <correlation set="x_idCorr" initiate="no" />
  </correlations>
</reply>
</sequence>
</process>

```

### Buyer service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:as="http://auction/auction.wsdl"
  xmlns:tns="http://buyer/buyer.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://buyer/buyer.wsdl">
  <wsdl:import namespace="http://auction/auction.wsdl"
    location="auction.wsdl" />
  <wsdl:portType name="initBuyerPT">
    <wsdl:operation name="init">
      <wsdl:input message="as:buyerReq" />
      <wsdl:output message="as:resp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="buyerBinding" type="as:buyerPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="answer">
      <soap:operation
        soapAction="http://auction/auction.wsdl/answer"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="initBuyerBinding"
    type="tns:initBuyerPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="init">
      <soap:operation
        soapAction="http://buyer/buyer.wsdl/init"
        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="buyerService">
    <wsdl:port name="buyerPort" binding="tns:buyerBinding">
      <soap:address location="http://localhost:8080
        /active-bpel/services/buyerService" />
    </wsdl:port>
  </wsdl:service>
  <wsdl:service name="initBuyerService">
    <wsdl:port name="initBuyerPort"
      binding="tns:initBuyerBinding">
      <soap:address location="http://localhost:8080
        /active-bpel/services/initBuyerService" />
    </wsdl:port>
  </wsdl:service>
  <plnk:partnerLinkType name="initBuyerPLT">
    <plnk:role name="initBuyer" portType="tns:initBuyerPT" />
  </plnk:partnerLinkType>
  <prop:property name="x_idProp" type="xsd:int" />
  <prop:propertyAlias propertyName="tns:x_idProp"
    messageType="as:clientAddressAUCTIONB"
    part="corrID" />
  <prop:propertyAlias propertyName="tns:x_idProp"
    messageType="as:resp"
    part="id" />

```

```

<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="as:buyerReq"
  part="id" />
</wsdl:definitions>

```

### Seller service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="sellerProcess"
  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://seller/seller.wsdl"
  xmlns:as="http://auction/auction.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://seller/seller.bpel">
  <import location="auction.wsdl"
    namespace="http://auction/auction.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <import location="seller.wsdl"
    namespace="http://seller/seller.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="auction"
      partnerLinkType="as:auctionSPLT"
      myRole="seller"
      partnerRole="auctionS" />
    <partnerLink name="initSellerPL"
      partnerLinkType="mwl:initSellerPLT"
      myRole="initSeller" />
  </partnerLinks>
  <variables>
    <variable name="var1" messageType="as:resp" />
    <variable name="var2"
      messageType="as:clientAddressAUCTIONS" />
    <variable name="var0" messageType="as:sellerReq" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr"
      properties="mwl:x_idProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>
    <receive partnerLink="initSellerPL" operation="init"
      variable="var0" createInstances="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="corrID" />
      </copy>
    </assign>
    </sequence>
    <invoke operation="submit" inputVariable="var0"
      partnerLink="auction">
      <correlations>
        <correlation set="x_idCorr"
          pattern="request" initiate="no" />
      </correlations>
    </invoke>
    <sequence>
      <assign>
        <copy>
          <from>
            <literal>
              <wsa:EndpointReference
                xmlns:wsa="http://schemas.xmlsoap.org/ws
                  /2003/03/addressing"
                xmlns:dyn="http://auction/auction.wsdl">
                <wsa:Address>http://localhost:8080
                  /active-bpel/services/sellerService

```

```

        </wsa:Address>
        <wsa:PortType>dyn:sellerPT</wsa:PortType>
        <wsa:ServiceName
            PortName="sellerServicePort">
            dyn:sellerService
        </wsa:ServiceName>
        </wsa:EndpointReference>
    </literal>
</from>
<to variable="var2" part="address" />
</copy>
</assign>
<invoke operation="setClientAddress"
    inputVariable="var2"
    partnerLink="auction">
    <correlations>
        <correlation set="x_idCorr"
            pattern="request"
            initiate="no" />
    </correlations>
</invoke>
</sequence>
<receive partnerLink="auction"
    operation="answer" variable="var1">
    <correlations>
        <correlation set="x_idCorr" initiate="no" />
    </correlations>
</receive>
<reply operation="init"
    partnerLink="initSellerPL" variable="var1">
    <correlations>
        <correlation set="x_idCorr" initiate="no" />
    </correlations>
</reply>
</sequence>
</process>

```

### Seller service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:as="http://auction/auction.wsdl"
    xmlns:tns="http://seller/seller.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://seller/seller.wsdl">
    <wsdl:import namespace="http://auction/auction.wsdl"
        location="auction.wsdl" />
    <wsdl:portType name="initSellerPT">
        <wsdl:operation name="init">
            <wsdl:input message="as:sellerReq" />
            <wsdl:output message="as:resp" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="initSellerBinding"
        type="tns:initSellerPT">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="init">
            <soap:operation
                soapAction="http://seller/seller.wsdl/init"
                style="document" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="sellerBinding"
        type="as:sellerPT">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="answer">
            <soap:operation
                soapAction="http://auction/auction.wsdl/answer"
                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="initSellerService">
    <wsdl:port name="initSellerPort"
        binding="tns:initSellerBinding">
        <soap:address
            location="http://localhost:8080/active-bpel
                /services/initSellerService" />
        </wsdl:port>
    </wsdl:service>
    <wsdl:service name="sellerService">
        <wsdl:port name="sellerPort"
            binding="tns:sellerBinding">
            <soap:address
                location="http://localhost:8080/active-bpel
                    /services/sellerService" />
            </wsdl:port>
        </wsdl:service>
    <plnk:partnerLinkType name="initSellerPLT">
        <plnk:role name="initSeller"
            portType="tns:initSellerPT" />
    </plnk:partnerLinkType>
    <prop:property name="x_idProp" type="xsd:int" />
    <prop:propertyAlias propertyName="tns:x_idProp"
        messageType="as:clientAddressAUCTIONS"
        part="corrID" />
    <prop:propertyAlias propertyName="tns:x_idProp"
        messageType="as:resp"
        part="id" />
    <prop:propertyAlias propertyName="tns:x_idProp"
        messageType="as:sellerReq"
        part="id" />
</wsdl:definitions>

```

### Auction service with compensation (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="auctionProcess"
    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://auction/auction.wsdl"
    xmlns:reg="http://register/register.wsdl"
    suppressJoinFailure="yes"
    targetNamespace="http://auction/auction.bpel">
    <import location="register.wsdl"
        namespace="http://register/register.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/" />
    <import location="auction.wsdl"
        namespace="http://auction/auction.wsdl"
        importType="http://schemas.xmlsoap.org/wsdl/" />
    <partnerLinks>
        <partnerLink name="auctionBPL"
            partnerLinkType="mwl:auctionBPLT"
            myRole="auctionB"
            partnerRole="buyer"
            initializePartnerRole="no" />
        <partnerLink name="registerComp"
            partnerLinkType="reg:registerCompPLT"
            partnerRole="registerComp" />
        <partnerLink name="register"
            partnerLinkType="reg:registerPLT"
            myRole="auction"
            partnerRole="register" />
        <partnerLink name="auctionSPL"
            partnerLinkType="mwl:auctionSPLT"
            myRole="auctionS"
            partnerRole="seller"
            initializePartnerRole="no" />
        <partnerLink name="auctionCancPL"
            partnerLinkType="mwl:auctionCancPLT"
            myRole="auctionCanc" />
    </partnerLinks>
    <variables>
        <variable name="var7"
            messageType="mwl:clientAddressAUCTIONS" />
        <variable name="var9"
            messageType="reg:clientAddressREGISTER" />
        <variable name="var0" messageType="mwl:sellerReq" />
        <variable name="var4" messageType="reg:unreg" />
        <variable name="var5" messageType="mwl:resp" />
        <variable name="var2" messageType="reg:regReq" />
        <variable name="var8"
            messageType="mwl:clientAddressAUCTIONB" />
        <variable name="var6" messageType="mwl:cancellation" />
        <variable name="var1" messageType="mwl:buyerReq" />
    </variables>

```



```

<variable name="var3" messageType="reg:regResp" />
</variables>
<correlationSets>
  <correlationSet name="x_auctionIdCorr"
    properties="mwl:x_auctionIdProp" />
</correlationSets>
<faultHandlers>
<catchAll>
  <sequence>
    <compensate />
    <sequence>
      <assign>
        <copy>
          <from>
            'The registration has been cancelled'
          </from>
          <to variable="var5" part="msg" />
        </copy>
      </assign>
      <flow>
        <invoke operation="answer"
          inputVariable="var5"
          partnerLink="auctionSPL">
          <correlations>
            <correlation set="x_auctionIdCorr"
              pattern="request"
              initiate="no" />
          </correlations>
        </invoke>
        <invoke operation="answer"
          inputVariable="var5"
          partnerLink="auctionBPL">
          <correlations>
            <correlation set="x_auctionIdCorr"
              pattern="request"
              initiate="no" />
          </correlations>
        </invoke>
      </flow>
    </sequence>
  </catchAll>
</faultHandlers>
<sequence>
  <flow>
    <sequence>
      <sequence>
        <receive partnerLink="auctionSPL"
          operation="submit"
          variable="var0"
          createInstance="yes">
          <correlations>
            <correlation set="x_auctionIdCorr"
              initiate="join" />
          </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>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var3" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var4" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var5" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var6" part="id" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var7" part="corrID" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var8" part="corrID" />
          </copy>
          <copy>
            <from variable="var0" part="id" />
            <to variable="var9" part="corrID" />
          </copy>
        </sequence>
      </sequence>
    </flow>
  </sequence>
</flow>
</sequence>
</catchAll>
</faultHandlers>
</sequence>

```

```

</assign>
</sequence>
<sequence>
  <receive partnerLink="auctionSPL"
    operation="setClientAddress"
    variable="var7">
    <correlations>
      <correlation set="x_auctionIdCorr"
        initiate="no" />
    </correlations>
  </receive>
  <assign>
    <copy>
      <from variable="var7" part="address" />
      <to partnerLink="auctionSPL" />
    </copy>
  </assign>
</sequence>
</sequence>
<sequence>
  <receive partnerLink="auctionBPL"
    operation="submit"
    variable="var1"
    createInstance="yes">
    <correlations>
      <correlation set="x_auctionIdCorr"
        initiate="join" />
    </correlations>
  </receive>
  <sequence>
    <receive partnerLink="auctionBPL"
      operation="setClientAddress"
      variable="var8">
      <correlations>
        <correlation set="x_auctionIdCorr"
          initiate="no" />
      </correlations>
    </receive>
    <assign>
      <copy>
        <from variable="var8" part="address" />
        <to partnerLink="auctionBPL" />
      </copy>
    </assign>
  </sequence>
</sequence>
</flow>
<assign>
  <copy>
    <from>1</from>
    <to variable="var2" part="amount" />
  </copy>
</assign>
<assign>
  <copy>
    <from>'Thank you!'</from>
    <to variable="var5" part="msg" />
  </copy>
</assign>
<scope>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <compensationHandler>
    <sequence>
      <assign>
        <copy>
          <from>'Auction cancelled'</from>
          <to variable="var4" part="motiv" />
        </copy>
      </assign>
      <invoke operation="unregister"
        inputVariable="var4"
        partnerLink="registerComp">
        <correlations>
          <correlation set="x_auctionIdCorr"
            pattern="request"
            initiate="no" />
        </correlations>
      </invoke>
    </sequence>
  </compensationHandler>
  <sequence>
    <invoke operation="process"
      inputVariable="var2"
      partnerLink="register">
    <correlations>
      <correlation set="x_auctionIdCorr"
    </correlations>
  </sequence>
</scope>

```

```

        pattern="request"
        initiate="no" />
</correlations>
</invoke>
<sequence>
  <assign>
    <copy>
      <from>
        <literal>
          <wsa:EndpointReference
            xmlns:wsa="http://schemas.xmlsoap.org
              /ws/2003/03/addressing"
            xmlns:dyn="http://register/register.wsdl">
          <wsa:Address>http://localhost:8080
            /active-bpel/services/auctionService
          </wsa:Address>
          <wsa:PortType>
            dyn:auctionPT
          </wsa:PortType>
          <wsa:ServiceName
            PortName="auctionServicePort">
            dyn:auctionService
          </wsa:ServiceName>
          </wsa:EndpointReference>
        </literal>
      </from>
      <to variable="var9" part="address" />
    </copy>
  </assign>
  <invoke operation="setClientAddress"
    inputVariable="var9"
    partnerLink="register">
    <correlations>
      <correlation set="x_auctionIdCorr"
        pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
</sequence>
<receive partnerLink="register"
  operation="regAnswer"
  variable="var3">
  <correlations>
    <correlation set="x_auctionIdCorr"
      initiate="no" />
  </correlations>
</receive>
<flow>
  <invoke operation="answer"
    inputVariable="var5"
    partnerLink="auctionSPL">
    <correlations>
      <correlation set="x_auctionIdCorr"
        pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
  <invoke operation="answer"
    inputVariable="var5"
    partnerLink="auctionBPL">
    <correlations>
      <correlation set="x_auctionIdCorr"
        pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
</flow>
</sequence>
</scope>
<receive partnerLink="auctionCancPL"
  operation="cancel"
  variable="var6">
  <correlations>
    <correlation set="x_auctionIdCorr"
      initiate="no" />
  </correlations>
</receive>
<throw xmlns:err="http://auction/error"
  faultName="err:auctionError" />
</sequence>
</process>

```

### Auction service with compensation (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws

```

```

    /2003/03/addressing"
    xmlns:reg="http://register/register.wsdl"
    xmlns:tns="http://auction/auction.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:plnk="http://docs.oasis-open.org/soap/
      /2.0/plnktype"
    xmlns:prop="http://docs.oasis-open.org/soap/
      /2.0/varprop"
    targetNamespace="http://auction/auction.wsdl">
  <wsdl:import namespace="http://register/register.wsdl"
    location="register.wsdl" />
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://auction/auction.wsdl">
    <xsd:import
      namespace="http://schemas.xmlsoap.org/ws
        /2003/03/addressing"
      schemaLocation="../schema/ws-addressing.xsd" />
    <xsd:element name="x_phoneNumberEL"
      type="xsd:string" />
    <xsd:element name="x_creditCardNumberBEL"
      type="xsd:string" />
    <xsd:element name="x_auctionIdEL"
      type="xsd:int" />
    <xsd:element name="x_respEL"
      type="xsd:string" />
    <xsd:element name="x_shippingCostEL"
      type="xsd:int" />
    <xsd:element name="x_creditCardNumberSEL"
      type="xsd:string" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="buyerReq">
    <wsdl:part name="id"
      element="tns:x_auctionIdEL" />
    <wsdl:part name="ccNum"
      element="tns:x_creditCardNumberBEL" />
    <wsdl:part name="phone"
      element="tns:x_phoneNumberEL" />
  </wsdl:message>
  <wsdl:message name="cancellation">
    <wsdl:part name="id"
      element="tns:x_auctionIdEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressAUCTIONB">
    <wsdl:part name="corrID"
      element="tns:x_auctionIdEL" />
    <wsdl:part name="address"
      element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:message name="resp">
    <wsdl:part name="id"
      element="tns:x_auctionIdEL" />
    <wsdl:part name="msg"
      element="tns:x_respEL" />
  </wsdl:message>
  <wsdl:message name="sellerReq">
    <wsdl:part name="id"
      element="tns:x_auctionIdEL" />
    <wsdl:part name="ccNum"
      element="tns:x_creditCardNumberSEL" />
    <wsdl:part name="cost"
      element="tns:x_shippingCostEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressAUCTIONS">
    <wsdl:part name="corrID"
      element="tns:x_auctionIdEL" />
    <wsdl:part name="address"
      element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:portType name="auctionCancPT">
    <wsdl:operation name="cancel">
      <wsdl:input message="tns:cancellation" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="sellerPT">
    <wsdl:operation name="answer">
      <wsdl:input message="tns:resp" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="auctionSPT">
    <wsdl:operation name="submit">
      <wsdl:input message="tns:sellerReq" />
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <wsdl:input
        message="tns:clientAddressAUCTIONS" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="buyerPT">
    <wsdl:operation name="answer">
      <wsdl:input message="tns:resp" />
    </wsdl:operation>
  </wsdl:portType>

```

```

</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="auctionBPT">
  <wsdl:operation name="submit">
    <wsdl:input message="tns:buyerReq" />
  </wsdl:operation>
  <wsdl:operation name="setClientAddress">
    <wsdl:input
      message="tns:clientAddressAUCTIONB" />
    </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="auctionCancBinding"
  type="tns:auctionCancPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org
      /soap/http" />
  <wsdl:operation name="cancel">
    <soap:operation
      soapAction="http://auction/auction.wsdl
        /cancel"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="auctionSBinding"
  type="tns:auctionSPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org
      /soap/http" />
  <wsdl:operation name="submit">
    <soap:operation
      soapAction="http://auction/auction.wsdl
        /submit"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="setClientAddress">
    <soap:operation
      soapAction="http://auction/auction.wsdl
        /setClientAddress"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="auctionBinding"
  type="reg:auctionPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org
      /soap/http" />
  <wsdl:operation name="regAnswer">
    <soap:operation
      soapAction="http://register/register.wsdl
        /regAnswer"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="auctionBBinding"
  type="tns:auctionBPT">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org
      /soap/http" />
  <wsdl:operation name="submit">
    <soap:operation
      soapAction="http://auction/auction.wsdl
        /submit"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
  <wsdl:operation
    name="setClientAddress">
    <soap:operation
      soapAction="http://auction/auction.wsdl
        /setClientAddress"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="auctionCancService">
  <wsdl:port name="auctionCancPort"
    binding="tns:auctionCancBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/auctionCancService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="auctionSService">
  <wsdl:port name="auctionSPort"
    binding="tns:auctionSBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/auctionSService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="auctionService">
  <wsdl:port name="auctionPort"
    binding="tns:auctionBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/auctionService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="auctionBService">
  <wsdl:port name="auctionBPort"
    binding="tns:auctionBBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/auctionBService" />
  </wsdl:port>
</wsdl:service>
<plnk:partnerLinkType name="auctionCancPLT">
  <plnk:role name="auctionCanc"
    portType="tns:auctionCancPT" />
</plnk:partnerLinkType>
<plnk:partnerLinkType name="auctionSPLT">
  <plnk:role name="auctionS"
    portType="tns:auctionSPT" />
  <plnk:role name="seller"
    portType="tns:sellerPT" />
</plnk:partnerLinkType>
<plnk:partnerLinkType name="auctionBPLT">
  <plnk:role name="auctionB"
    portType="tns:auctionBPT" />
  <plnk:role name="buyer"
    portType="tns:buyerPT" />
</plnk:partnerLinkType>
<prop:property name="x_auctionIdProp"
  type="xsd:int" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="reg:clientAddressREGISTER"
  part="corrID" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:clientAddressAUCTIONB"
  part="corrID" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:clientAddressAUCTIONS"
  part="corrID" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:cancellation"
  part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:resp"
  part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="reg:unreg" part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="reg:regResp" part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="reg:regReq" part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:buyerReq" part="id" />
<prop:propertyAlias
  propertyName="tns:x_auctionIdProp"
  messageType="tns:sellerReq" part="id" />
</wsdl:definitions>
<?xml version="1.0" encoding="UTF-8"?>
<process name="shipping_serviceProcess"

```

### Shipping service (WS-BPEL file)

```

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:bck="http://example/backendService
/backend_service.wsdl"
xmlns:mwl="http://example/shipping_service.wsdl"
suppressJoinFailure="yes"
targetNamespace="http://example
/shipping_service.bpel"
<import location="shipping_service.wsdl"
namespace="http://example/shipping_service.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
<import location="backend_service.wsdl"
namespace="http://example/backendService
/backend_service.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
<partnerLinks>
<partnerLink name="backend"
partnerLinkType="bck:clientPLT"
partnerRole="p_backend" />
<partnerLink name="custPL"
partnerLinkType="mwl:custPLT"
myRole="p_ship"
partnerRole="x_cust"
initializePartnerRole="no" />
</partnerLinks>
<variables>
<variable name="var0"
messageType="mwl:shipOrder" />
<variable name="var3"
messageType="mwl:shippingErrorMsg" />
<variable name="var7"
messageType="mwl:clientAddressP_SHIP" />
<variable name="var5"
messageType="bck:number" />
<variable name="var4"
messageType="bck:id" />
<variable name="var1"
messageType="mwl:shippingNoticeMsg" />
<variable name="var2"
messageType="mwl:shippingNoticeMsg" />
<variable name="var6"
type="xsd:integer" />
</variables>
<correlationSets>
<correlationSet name="x_idCorr"
properties="mwl:x_idProp" />
</correlationSets>
<faultHandlers>
<catchAll>
<sequence>
<compensate />
<empty />
</sequence>
</catchAll>
</faultHandlers>
<sequence>
<sequence>
<receive partnerLink="custPL"
operation="o_req"
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>
<copy>
<from variable="var0" part="id" />
<to variable="var3" part="id" />
</copy>
<copy>
<from variable="var0" part="id" />
<to variable="var4" part="id" />
</copy>
<copy>
<from variable="var0" part="id" />
<to variable="var5" part="id" />
</copy>
<copy>
<from variable="var0" part="id" />
<to variable="var7" part="corrID" />
</copy>
</sequence>
</sequence>
</sequence>
<receive partnerLink="custPL"
operation="setClientAddress"
variable="var7">
<correlations>
<correlation set="x_idCorr"
initiate="no" />
</correlations>
</receive>
<assign>
<copy>
<from variable="var7" part="address" />
<to partnerLink="custPL" />
</copy>
</assign>
</sequence>
<if>
<condition>$var0.shipComplete &gt; 0</condition>
<invoke operation="o_notice"
inputVariables="var1"
partnerLink="custPL">
<correlations>
<correlation set="x_idCorr"
pattern="request"
initiate="no" />
</correlations>
</invoke>
<else>
<scope>
<faultHandlers>
<catchAll>
<sequence>
<compensate />
<sequence>
<assign>
<copy>
<from>'Sorry, the required item
is out of stock '</from>
<to variable="var3"
part="errorMsg" />
</copy>
</assign>
<invoke operation="o_err"
inputVariable="var3"
partnerLink="custPL">
<correlations>
<correlation set="x_idCorr"
pattern="request"
initiate="no" />
</correlations>
</invoke>
</sequence>
</catchAll>
</faultHandlers>
<compensationHandler>
<empty />
</compensationHandler>
<sequence>
<assign>
<copy>
<from>0</from>
<to variable="var6" />
</copy>
</assign>
<while>
<condition>
$var6 &lt; $var0.items
</condition>
<sequence>
<sequence>
<invoke operation="o_num"
inputVariable="var4"
partnerLink="backend"
outputVariable="var5">
<correlations>
<correlation set="x_idCorr"
pattern="request-response"
initiate="no" />
</correlations>
</invoke>
<assign>
<copy>
<from variable="var5"
part="num" />
<to variable="var2"

```

```

        part="items" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var0"
    part="id" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var1"
    part="id" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var2"
    part="id" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var3"
    part="id" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var4"
    part="id" />
</copy>
<copy>
  <from variable="var5"
    part="id" />
  <to variable="var7"
    part="corrID" />
</copy>
</assign>
</sequence>
<if>
  <condition>
    $var2.items &lt;= 0
  </condition>
  <throw
    xmlns:err="http://example/error"
    faultName="err:shipping_serviceError" />
</else>
  <sequence>
    <invoke operation="o_notice"
      inputVariable="var2"
      partnerLink="custPL">
      <correlations>
        <correlation set="x_idCorr"
          pattern="request"
          initiate="no" />
      </correlations>
    </invoke>
    <assign>
      <copy>
        <from>$var6 + $var2.items</from>
        <to variable="var6" />
      </copy>
    </assign>
  </sequence>
</else>
</if>
</sequence>
</while>
</sequence>
</scope>
</else>
</if>
</sequence>
</process>

```

### Shipping service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://schemas.xmlsoap.org/ws
    /2003/03/addressing"
  xmlns:bck="http://example/backendService
    /backend_service.wsdl"
  xmlns:tns="http://example
    /shipping_service.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

```

```

  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:plnk="http://docs.oasis-open.org/soap
    /2.0/plnktype"
  xmlns:prop="http://docs.oasis-open.org/soap
    /2.0/varprop"
  targetNamespace="http://example
    /shipping_service.wsdl">
  <wsdl:import
    namespace="http://example/backendService
    /backend_service.wsdl"
    location="backend_service.wsdl" />
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://example
        /shipping_service.wsdl">
      <xsd:import
        namespace="http://schemas.xmlsoap.org/ws
          /2003/03/addressing"
        schemaLocation="../schema/ws-addressing.xsd" />
      <xsd:element name="x_countEL" type="xsd:int" />
      <xsd:element name="x_idEL" type="xsd:int" />
      <xsd:element name="x_itemsEL" type="xsd:int" />
      <xsd:element name="x_sorryEL" type="xsd:string" />
      <xsd:element name="x_cEL" type="xsd:int" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="shippingNoticeMsg">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="items" element="tns:x_itemsEL" />
  </wsdl:message>
  <wsdl:message name="clientAddressP_SHIP">
    <wsdl:part name="corrID" element="tns:x_idEL" />
    <wsdl:part name="address"
      element="wsa:EndpointReference" />
  </wsdl:message>
  <wsdl:message name="shippingErrorMsg">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="errorMsg"
      element="tns:x_sorryEL" />
  </wsdl:message>
  <wsdl:message name="shipOrder">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="shipComplete"
      element="tns:x_cEL" />
    <wsdl:part name="items"
      element="tns:x_itemsEL" />
  </wsdl:message>
  <wsdl:portType name="x_custPT">
    <wsdl:operation name="o_notice">
      <wsdl:input
        message="tns:shippingNoticeMsg" />
    </wsdl:operation>
    <wsdl:operation name="o_err">
      <wsdl:input
        message="tns:shippingErrorMsg" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="p_shipPT">
    <wsdl:operation name="o_req">
      <wsdl:input message="tns:shipOrder" />
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <wsdl:input message="tns:clientAddressP_SHIP" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="p_shipBinding" type="tns:p_shipPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="o_req">
      <soap:operation
        soapAction="http://example/shipping_service.wsdl
          /o_req"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="setClientAddress">
      <soap:operation
        soapAction="http://example/shipping_service.wsdl
          /setClientAddress"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="p_shipService">
    <wsdl:port name="p_shipPort"
      binding="tns:p_shipBinding">
      <soap:address
        location="http://localhost:8080/active-bpel
          /services/p_shipService" />
    </wsdl:port>
  </wsdl:service>

```

```

</wsdl:port>
</wsdl:service>
<plnk:partnerLinkType name="custPLT">
  <plnk:role name="p_ship" portType="tns:p_shipPT" />
  <plnk:role name="x_cust" portType="tns:x_custPT" />
</plnk:partnerLinkType>
<prop:property name="x_idProp" type="xsd:int" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:clientAddressP_SHIP" part="corrID" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="bck:number" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="bck:id" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:shippingErrorMsg" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:shippingNoticeMsg" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:shippingNoticeMsg" part="id" />
<prop:propertyAlias propertyName="tns:x_idProp"
  messageType="tns:shipOrder" part="id" />
</wsdl:definitions>

```

```

<receive partnerLink="p_humanPL"
  operation="o_humanInteraction"
  variable="var1">
  <correlations>
    <correlation set="x_idCorr" initiate="no" />
  </correlations>
</receive>
<assign>
  <copy>
    <from variable="var1" part="num" />
    <to variable="var2" part="num" />
  </copy>
</assign>
</sequence>
<reply operation="o_num"
  partnerLink="clientPL"
  variable="var2">
  <correlations>
    <correlation set="x_idCorr" initiate="no" />
  </correlations>
</reply>
</sequence>
</process>

```

### Back-end service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="backend_serviceProcess"
  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/backendService
    /backend_service.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://example/backendService
    /backend_service.bpel">
  <import location="backend_service.wsdl"
    namespace="http://example/backendService
    /backend_service.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="p_humanPL"
      partnerLinkType="mwl:p_humanPLT"
      myRole="p_human" />
    <partnerLink name="clientPL"
      partnerLinkType="mwl:clientPLT"
      myRole="p_backend" />
  </partnerLinks>
  <variables>
    <variable name="var2" messageType="mwl:number" />
    <variable name="var1" messageType="mwl:human" />
    <variable name="var0" messageType="mwl:id" />
  </variables>
  <correlationSets>
    <correlationSet name="x_idCorr"
      properties="mwl:x_idProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>
    <sequence>
      <receive partnerLink="clientPL"
        operation="o_num"
        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>
  </sequence>

```

### Back-end service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example/backendService
    /backend_service.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/backendService
    /backend_service.wsdl">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://example/backendService
        /backend_service.wsdl">
      <xsd:element name="x_idEL" type="xsd:int" />
      <xsd:element name="x_numEL" type="xsd:int" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="id">
    <wsdl:part name="id" element="tns:x_idEL" />
  </wsdl:message>
  <wsdl:message name="human">
    <wsdl:part name="id" element="tns:x_idEL" />
    <wsdl:part name="num" element="tns:x_numEL" />
  </wsdl:message>
  <wsdl:message name="number">
    <wsdl:part name="num" element="tns:x_numEL" />
    <wsdl:part name="id" element="tns:x_idEL" />
  </wsdl:message>
  <wsdl:portType name="p_backendPT">
    <wsdl:operation name="o_num">
      <wsdl:input message="tns:id" />
      <wsdl:output message="tns:number" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:portType name="p_humanPT">
    <wsdl:operation name="o_humanInteraction">
      <wsdl:input message="tns:human" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="p_backendBinding"
    type="tns:p_backendPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="o_num">
      <soap:operation
        soapAction="http://example/backendService
          /backend_service.wsdl/o_num"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="p_humanBinding"
    type="tns:p_humanPT">

```

```

<soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="o_humanInteraction">
  <soap:operation
    soapAction="http://example/backendService
      /backend_service.wsdl/o_humanInteraction"
    style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="p_backendService">
  <wsdl:port name="p_backendPort"
    binding="tns:p_backendBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/p_backendService" />
  </wsdl:port>
</wsdl:service>
<wsdl:service name="p_humanService">
  <wsdl:port name="p_humanPort"
    binding="tns:p_humanBinding">
    <soap:address
      location="http://localhost:8080/active-bpel
        /services/p_humanService" />
  </wsdl:port>
</wsdl:service>
<plnk:partnerLinkType name="clientPLT">
  <plnk:role name="p_backend"
    portType="tns:p_backendPT" />
</plnk:partnerLinkType>
<plnk:partnerLinkType name="p_humanPLT">
  <plnk:role name="p_human" portType="tns:p_humanPT" />
</plnk:partnerLinkType>
<prop:property name="x_idProp" type="xsd:int" />
<prop:propertyAlias
  propertyName="tns:x_idProp"
  messageType="tns:number" part="id" />
<prop:propertyAlias
  propertyName="tns:x_idProp"
  messageType="tns:human" part="id" />
<prop:propertyAlias
  propertyName="tns:x_idProp"
  messageType="tns:id" part="id" />
</wsdl:definitions>

```

```

</variables>
<correlationSets>
  <correlationSet name="y_idCorr"
    properties="mwl:y_idProp" />
</correlationSets>
<faultHandlers>
  <catchAll>
    <sequence>
      <compensate />
      <empty />
    </sequence>
  </catchAll>
</faultHandlers>
<sequence>
  <sequence>
    <receive partnerLink="cltPL"
      operation="o_init"
      variable="var0"
      createInstances="yes">
      <correlations>
        <correlation set="y_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>
      <copy>
        <from variable="var0" part="id" />
        <to variable="var4" part="id" />
      </copy>
      <copy>
        <from variable="var0" part="id" />
        <to variable="var5" part="corrID" />
      </copy>
    </assign>
  </sequence>
  <assign>
    <copy>
      <from>concat('ORDER: ',string($var0.id),
        ' BEGIN ')</from>
      <to variable="var4" part="resp" />
    </copy>
  </assign>
  <invoke operation="o_req"
    inputVariable="var0"
    partnerLink="srv">
    <correlations>
      <correlation set="y_idCorr"
        pattern="request"
        initiate="no" />
    </correlations>
  </invoke>
</sequence>
<assign>
  <copy>
    <from>
      <literal>
        <wsa:EndpointReference
          xmlns:wsa="http://schemas.xmlsoap.org
            /ws/2003/03/addressing"
          xmlns:dyn="http://example
            /shipping_service.wsdl">
          <wsa:Address>http://localhost:8080
            /active-bpel/services
            /x_custService
          </wsa:Address>
          <wsa:PortType>
            dyn:x_custPT
          </wsa:PortType>
          <wsa:ServiceName>
            PortName="x_custServicePort">
            dyn:x_custService
          </wsa:ServiceName>
        </wsa:EndpointReference>
      </literal>
    </from>
    <to variable="var5" part="address" />
  </copy>
</assign>
  <invoke operation="setClientAddress"
    inputVariable="var5"
    partnerLink="srv">
    <correlations>
      <correlation set="y_idCorr"
        pattern="request"
        initiate="no" />

```

### Client service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="shipping_clientProcess"
  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/shipping_client.wsdl"
  xmlns:shs="http://example/shipping_service.wsdl"
  suppressJoinFailure="yes"
  targetNamespace="http://example/shipping_client.bpel">
<import location="shipping_service.wsdl"
  namespace="http://example/shipping_service.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/" />
<import location="shipping_client.wsdl"
  namespace="http://example/shipping_client.wsdl"
  importType="http://schemas.xmlsoap.org/wsdl/" />
<partnerLinks>
  <partnerLink name="cltPL"
    partnerLinkType="mwl:cltPLT"
    myRole="p_init" />
  <partnerLink name="srv"
    partnerLinkType="shs:custPLT"
    myRole="x_cust"
    partnerRole="p_ship" />
</partnerLinks>
<variables>
  <variable name="var1"
    messageType="shs:shippingErrorMsg" />
  <variable name="var3"
    type="xsd:integer" />
  <variable name="var4"
    messageType="mwl:response" />
  <variable name="var0"
    messageType="shs:shipOrder" />
  <variable name="var5"
    messageType="shs:clientAddressP_SHIP" />
  <variable name="var2"
    messageType="shs:shippingNoticeMsg" />

```

```

    </correlations>
  </invoke>
</sequence>
<assign>
  <copy>
    <from>0</from>
    <to variable="var3" />
  </copy>
</assign>
<while>
  <condition>$var3 &lt; $var0.items</condition>
  <pick>
    <onMessage partnerLink="srv"
      operation="o_notice"
      variable="var2">
      <correlations>
        <correlation set="y_idCorr"
          initiate="no" />
      </correlations>
      <sequence>
        <assign>
          <copy>
            <from>$var3 + $var2.items</from>
            <to variable="var3" />
          </copy>
        </assign>
        <assign>
          <copy>
            <from>concat(string($var4.resp),
              'NOTICE: sent items=',
              string($var2.items),'; ' )
            </from>
            <to variable="var4" part="resp" />
          </copy>
        </assign>
      </sequence>
    </onMessage>
    <onMessage partnerLink="srv"
      operation="o_err"
      variable="var1">
      <correlations>
        <correlation set="y_idCorr"
          initiate="no" />
      </correlations>
      <sequence>
        <assign>
          <copy>
            <from>$var0.items</from>
            <to variable="var3" />
          </copy>
        </assign>
        <assign>
          <copy>
            <from>
              concat(string($var4.resp),
                'ERROR: ',
                string($var1.errorMessage)
              )
            </from>
            <to variable="var4" part="resp" />
          </copy>
        </assign>
      </sequence>
    </onMessage>
  </pick>
</while>
<assign>
  <copy>
    <from>
      concat(string($var4.resp), ' END')
    </from>
    <to variable="var4" part="resp" />
  </copy>
</assign>
<reply operation="o_init"
  partnerLink="cltPL"
  variable="var4">
  <correlations>
    <correlation set="y_idCorr"
      initiate="no" />
  </correlations>
</reply>
</sequence>
</process>

```

### Client service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wSDL:definitions

```

```

  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:shs="http://example/shipping_service.wSDL"
  xmlns:tns="http://example/shipping_client.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/shipping_client.wSDL">
  <wSDL:import
    namespace="http://example/shipping_service.wSDL"
    location="shipping_service.wSDL" />
  <wSDL:types>
    <xsd:schema
      targetNamespace="http://example
        /shipping_client.wSDL">
      <xsd:element name="y_respEL" type="xsd:string" />
      <xsd:element name="y_idEL" type="xsd:int" />
    </xsd:schema>
  </wSDL:types>
  <wSDL:message name="response">
    <wSDL:part name="id" element="tns:y_idEL" />
    <wSDL:part name="resp" element="tns:y_respEL" />
  </wSDL:message>
  <wSDL:portType name="p_initPT">
    <wSDL:operation name="o_init">
      <wSDL:input message="shs:shipOrder" />
      <wSDL:output message="tns:response" />
    </wSDL:operation>
  </wSDL:portType>
  <wSDL:binding name="x_custBinding" type="shs:x_custPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wSDL:operation name="o_notice">
      <soap:operation
        soapAction="http://example/shipping_service.wSDL
          /o_notice"
        style="document" />
      <wSDL:input>
        <soap:body use="literal" />
      </wSDL:input>
    </wSDL:operation>
    <wSDL:operation name="o_err">
      <soap:operation
        soapAction="http://example/shipping_service.wSDL
          /o_err"
        style="document" />
      <wSDL:input>
        <soap:body use="literal" />
      </wSDL:input>
    </wSDL:operation>
  </wSDL:binding>
  <wSDL:binding name="p_initBinding" type="tns:p_initPT">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wSDL:operation name="o_init">
      <soap:operation
        soapAction="http://example/shipping_client.wSDL
          /o_init"
        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="x_custService">
    <wSDL:port name="x_custPort"
      binding="tns:x_custBinding">
      <soap:address
        location="http://localhost:8080/active-bpel
          /services/x_custService" />
    </wSDL:port>
  </wSDL:service>
  <wSDL:service name="p_initService">
    <wSDL:port name="p_initPort"
      binding="tns:p_initBinding">
      <soap:address
        location="http://localhost:8080/active-bpel
          /services/p_initService" />
    </wSDL:port>
  </wSDL:service>
  <plnk:partnerLinkType name="cltPLT">
    <plnk:role name="p_init" portType="tns:p_initPT" />
  </plnk:partnerLinkType>
  <prop:property name="y_idProp" type="xsd:int" />
  <prop:propertyAlias
    propertyName="tns:y_idProp"
    messageType="shs:clientAddressP_SHIP" part="corrID" />
  <prop:propertyAlias

```



```

        propertyName="tns:y_idProp"
        messageType="tns:response" part="id" />
<prop:propertyAlias
  propertyName="tns:y_idProp"
  messageType="shs:shippingNoticeMsg" part="id" />
<prop:propertyAlias
  propertyName="tns:y_idProp"
  messageType="shs:shippingErrorMsg" part="id" />
<prop:propertyAlias
  propertyName="tns:y_idProp"
  messageType="shs:shipOrder" part="id" />
</wsdl:definitions>

```

## Weather and temperature service (WS-BPEL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<process name="zipWeatherClientProcess"
  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:tem="http://webservices.daehosting.com/temperature"
  xmlns:mwl="http://zipWeather/zipWeatherClient.wsdl"
  xmlns:wth="http://ws.cdyne.com/WeatherWS/"
  suppressJoinFailure="yes"
  targetNamespace="http://zipWeather/zipWeatherClient.bpel">
  <import location="Weather.wsdl"
    namespace="http://ws.cdyne.com/WeatherWS/"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <import location="zipWeatherClient.wsdl"
    namespace="http://zipWeather/zipWeatherClient.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <import location="Converter.wsdl"
    namespace="http://webservices.daehosting.com/temperature"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks>
    <partnerLink name="initWeatherClientPL"
      partnerLinkType="mwl:initWeatherClientPLT"
      myRole="initWeatherClient" />
    <partnerLink name="weatherPL"
      partnerLinkType="mwl:weatherPLT"
      partnerRole="role0" />
    <partnerLink name="converterPL"
      partnerLinkType="mwl:converterPLT"
      partnerRole="role1" />
  </partnerLinks>
  <variables>
    <variable name="var3" messageType="mwl:resp" />
    <variable name="var2" messageType="mwl:req" />
    <variable name="var7"
      messageType="tem:FahrenheitToCelciusSoapResponse" />
    <variable name="var8" type="xsd:string" />
    <variable name="var6"
      messageType="tem:FahrenheitToCelciusSoapRequest" />
    <variable name="var4" type="xsd:string" />
    <variable name="var0"
      messageType="wth:GetCityWeatherByZIPSoapIn" />
    <variable name="var5" type="xsd:decimal" />
    <variable name="var1"
      messageType="wth:GetCityWeatherByZIPSoapOut" />
  </variables>
  <correlationSets>
    <correlationSet name="numCorr"
      properties="mwl:numProp" />
  </correlationSets>
  <faultHandlers>
    <catchAll>
      <sequence>
        <compensate />
        <empty />
      </sequence>
    </catchAll>
  </faultHandlers>
  <sequence>
    <sequence>
      <receive partnerLink="initWeatherClientPL"
        operation="init"
        variable="var2"
        createInstance="yes">
        <correlations>
          <correlation set="numCorr" initiate="yes" />
        </correlations>
      </receive>
      <assign>

```

```

        <copy>
          <from variable="var2" part="init" />
          <to variable="var3" part="init" />
        </copy>
        <copy>
          <from variable="var2" part="request" />
          <to variable="var0" part="parameters" />
        </copy>
      </assign>
    </sequence>
    <invoke operation="GetCityWeatherByZIP"
      inputVariable="var0"
      partnerLink="weatherPL"
      outputVariable="var1" />
    <assign>
      <copy>
        <from variable="var1" part="parameters">
          <query>/wth:GetCityWeatherByZIPResponse
            /wth:GetCityWeatherByZIPResult
            /wth:Description</query>
        </from>
        <to variable="var3" part="response" />
      </copy>
    </assign>
    <assign>
      <copy>
        <from variable="var1" part="parameters">
          <query>/wth:GetCityWeatherByZIPResponse
            /wth:GetCityWeatherByZIPResult
            /wth:City</query>
        </from>
        <to variable="var4" />
      </copy>
    </assign>
    <assign>
      <copy>
        <from variable="var1" part="parameters">
          <query>/wth:GetCityWeatherByZIPResponse
            /wth:GetCityWeatherByZIPResult
            /wth:Temperature</query>
        </from>
        <to variable="var5" />
      </copy>
    </assign>
    <assign>
      <copy>
        <from>concat('The current weather at ',
          string($var4), ' is: ',
          string($var3.response),
          ', the temperature is: ')</from>
        <to variable="var3" part="response" />
      </copy>
    </assign>
    <if>
      <condition>$var2.scale = 'c'</condition>
      <sequence>
        <assign>
          <copy>
            <from>
              <literal>
                <tem:FahrenheitToCelcius>
                  <tem:nFahrenheit>0</tem:nFahrenheit>
                </tem:FahrenheitToCelcius>
              </literal>
            </from>
            <to variable="var6" part="parameters" />
          </copy>
        </assign>
        <assign>
          <copy>
            <from>$var5</from>
            <to variable="var6" part="parameters">
              <query>/tem:FahrenheitToCelcius
                /tem:nFahrenheit</query>
            </to>
          </copy>
        </assign>
        <invoke operation="FahrenheitToCelcius"
          inputVariable="var6"
          partnerLink="converterPL"
          outputVariable="var7" />
        <assign>
          <copy>
            <from variable="var7" part="parameters">
              <query>/tem:FahrenheitToCelciusResponse
                /tem:FahrenheitToCelciusResult</query>
            </from>
            <to variable="var8" />
          </copy>
        </assign>
        <assign>
          <copy>
            <from>concat(string($var3.response),

```

```

                string($var8), ' rC')</from>
            <to variable="var3" part="response" />
        </copy>
    </assign>
</sequence>
<else>
    <assign>
        <copy>
            <from>concat(string($var3.response),
                string($var5), ' rF')</from>
            <to variable="var3" part="response" />
        </copy>
    </assign>
</else>
</if>
<reply operation="init"
    partnerLink="initWeatherClientPL"
    variable="var3">
    <correlations>
        <correlation set="numCorr" initiate="no" />
    </correlations>
</reply>
</sequence>
</process>

```

```

    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="initWeatherClientService">
    <wsdl:port name="initWeatherClientPort"
        binding="tns:initWeatherClientBinding">
        <soap:address
            location="http://localhost:8080/active-bpel
                /services/initWeatherClientService" />
        </wsdl:port>
    </wsdl:service>
    <plnk:partnerLinkType name="converterPLT">
        <plnk:role name="role1"
            portType="tem:TemperatureConversionsSoapType" />
    </plnk:partnerLinkType>
    <plnk:partnerLinkType name="weatherPLT">
        <plnk:role name="role0" portType="wth:WeatherSoap" />
    </plnk:partnerLinkType>
    <plnk:partnerLinkType name="initWeatherClientPLT">
        <plnk:role name="initWeatherClient"
            portType="tns:initWeatherClientPT" />
    </plnk:partnerLinkType>
    <prop:property name="numProp" type="xsd:integer" />
    <prop:propertyAlias
        propertyName="tns:numProp"
        messageType="tns:resp" part="init" />
    <prop:propertyAlias
        propertyName="tns:numProp"
        messageType="tns:req" part="init" />
</wsdl:definitions>

```

## Weather and temperature service (WSDL file)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tem="http://webservices.daehosting.com
        /temperature"
    xmlns:wth="http://ws.cdyne.com/WeatherWS/"
    xmlns:tns="http://zipWeather/zipWeatherClient.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://zipWeather
        /zipWeatherClient.wsdl">
    <wsdl:import
        namespace="http://webservices.daehosting.com
            /temperature"
        location="Converter.wsdl" />
    <wsdl:import namespace="http://ws.cdyne.com/WeatherWS/"
        location="Weather.wsdl" />
    <wsdl:types>
        <xsd:schema
            targetNamespace="http://zipWeather
                /zipWeatherClient.wsdl">
            <xsd:element name="scaleEL" type="xsd:string" />
            <xsd:element name="numEL" type="xsd:integer" />
            <xsd:element name="xEL" type="xsd:string" />
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="req">
        <wsdl:part name="init" element="tns:numEL" />
        <wsdl:part name="scale" element="tns:scaleEL" />
        <wsdl:part name="request"
            element="wth:GetCityWeatherByZIP" />
    </wsdl:message>
    <wsdl:message name="resp">
        <wsdl:part name="init" element="tns:numEL" />
        <wsdl:part name="response" element="tns:xEL" />
    </wsdl:message>
    <wsdl:portType name="initWeatherClientPT">
        <wsdl:operation name="init">
            <wsdl:input message="tns:req" />
            <wsdl:output message="tns:resp" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="initWeatherClientBinding"
        type="tns:initWeatherClientPT">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="init">
            <soap:operation
                soapAction="http://zipWeather
                    /zipWeatherClient.wsdl/init"
                style="document" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

```