Towards Adaptability Support in Collaborative Business Processes

Ismail Khriss
Université du Québec à Rimouski
Département d'informatique, de mathématiques et de génie
300, allée des Ursulines
Rimouski (QC) G5L 3A1 Canada
ismail_khriss@uqar.qc.ca

Enrico Lévesque
Université du Québec à Montréal
Département d'informatique
C.P. 8888, Succ. Centre-Ville
Montréal (QC) H3C 3P8 Canada
levesque.enrico@courrier.uqam.ca

Guy Tremblay
Université du Québec à Montréal,
Département d'informatique
C.P. 8888, Succ. Centre-Ville
Montréal (QC) H3C 3P8 Canada
tremblay.guy@uqam.ca

André Jacques
Université du Québec à Rimouski
Département d'informatique, de mathématiques et de génie
300, allée des Ursulines
Rimouski (QC) G5L 3A1 Canada
andre_jacque@uqar.qc.ca

Abstract

Service-oriented architecture (SOA) brings new perspectives not only to software architecture but also to enterprise business processes. SOA promotes the use of loosely coupled services to automate business processes. The automation of business processes raises several challenges for enterprises. One of those challenges relates to the maintenance of business processes, more precisely how to facilitate changes within existing business processes, possibly even at run-time. Furthermore, changing a collaborative business process can have an impact on the contract specified between the parties involved. Thus, a business process may need to adapt to meet a new contract.

Some approaches propose solutions to support business process adaptability; however, they do not handle adaptability in the context of multi-partner collaborative business processes. In this paper, we propose a new approach to support adaptability for such collaborative processes. This approach uses a protocol, called Change Protocol for Collaboration (CPC), for managing the changes that can have incidence on the contract.

1 Introduction

Service-oriented architecture (SOA) brings new perspectives not only to software architecture but also to enterprise business processes. SOA promotes the use of loosely coupled services to automate business processes. The automation of business processes raises several challenges for enterprises. The various parts played by the business departments and/or partners make the processes more complex. Complexity comes from the number of actions necessary to carry out a business process, the number of implied roles, the inter-dependencies between those roles, and their continual evolution. One of those challenges relates to the maintenance of business processes within multi-partner collaborations, more precisely how to facilitate changes within existing multi-partner processes, possibly even at run-time.

Complex multi-partner collaborations require defining some appropriate collaboration contract or agreement between the partners involved. Such collaboration contracts can be expressed in various ways, e.g., UML sequence or collaboration diagrams, WS-CDL choreography description [20], etc. These collaboration agreements may also be changed, changes possibly initiated by one of the partners. There are two main models of how messages are exchanged between business partners. The first is when messages
follow standards, such as EDI (Electronic Data Interchange) transactions [2]. The second is when one of the partners imposes its own standard. The latter model generally follows the Master-Slave pattern. The Master (usually a large business) decides on the type of exchanges and all Slaves (usually small businesses) must adapt their business processes to do business with the Master. In addition, even when the messages exchanged between business partners follow a standard such as EDI, a partner could still decide that some optional fields of a transaction (e.g., 810 invoice transaction) are mandatory because of new business rules.

Various approaches propose solutions to support business process adaptability; however, they do not handle adaptability in the context of multi-partner collaborative business processes. In this paper, we propose a new approach to support adaptability for such processes. This approach uses a new protocol, called Change Protocol for Collaboration (CPC), for managing the changes that can have incidences on the collaboration contract. We propose also a new algorithm for supporting migration of running business process instances to their new schemas. This migration is considered one of the major issues of business process adaptability.

The rest of the paper is organized as follows. First, we introduce the key challenges of business processes adaptability. Then, we give an overview of the proposed protocol and present the algorithm for supporting migration of process instances. Next, we discuss related work. Finally, we conclude and discuss possible future work.

2 Business Process Adaptability

In this section, we first give a classification of business process changes. Then, we present some examples of changes. Finally, we discuss issues related with the support of adaptive business processes.

2.1 Classification of changes

Changes to business processes can be classified along three dimensions: Time, Scope and Perspective (the perspective dimension is inspired from [1]). Time relates to whether changes are made at design-time or run-time. Scope describes the granularity of changes: i) Changes can be made to an overall process schema, as part of continuous improvement within the business process structure, in which case this would impact all process instances; ii) Changes may concern only a few specific cases, e.g., an exception to a specialized transaction.

Along the Perspective dimension, we identify three possible perspectives: Process, Data, and Service. The Process perspective describes the control flow aspect of business processes. Activities can be added, deleted, their ordering can be changed, etc. The Data perspective describes variables—such as messages exchanged between partners or intermediate data used in business logic to compose messages—, data expressions used for control purposes, and correlation data used for routing messages to appropriate process instances. Data can be added, deleted or changed. The Service perspective describes the services bound to processes. Changes to a service can pertain to its type (use of another service), or its provider (doing business with another partner). Note that various business process management systems (BPMS), depending on business rules and instance data, make it possible to select a Web service provider at run-time (see Section 5). We call this dynamic binding since the service port type is bound to the selected provided port’s address at run-time.

2.2 Examples of changes

To illustrate process changes, we introduce a simple business scenario. A Buyer wants to perform a purchase order. After receiving the PORequest, the Seller asks for a credit authorization from the FinancialInst. If the authorization is denied, the order is canceled. Otherwise, processing proceeds by making a shipping order to the Shipper. Then, the Buyer is informed of the order status. This business scenario is presented in Fig. 1 (UML sequence diagram). The arrows between actors correspond to exchanged messages. In our approach, we use the Web Services Choreography Description Language (WS-CDL) [20] for describing multi-party collaborations from a global viewpoint; however, for simplicity of presentation, the paper uses UML diagrams and scenarios.

Fig. 2 presents the local view of the Seller business process (UML activity diagram). Note that the process is completed by internal activities such as Process Order and Invoicing. We use the Business Process Execution Language for Web Services (WS-BPEL) [15] for describing (local) business process orchestrations.

In Fig. 3 (a BPEL description of this process is given at the end of this paper), two activities (gray filled) have been added, Check Inventory and BackOrder Response, as well as a decision (new data expression) on whether to proceed or not with the
Fig. 1. A business scenario

Fig. 2. Local process for Seller

Fig. 3. Modification of the Seller process
Then, we modified the workflow after Process Order to execute concurrently with the Invoicing activity as well as with the Order To Shipper and Receive Shipping Order Response activities. We want these changes to be applied to all running instances (Time and Scope dimensions). Note that the newly added activities result from a new kind of message (BackOrder Response) exchanged between the seller and the buyer and, therefore, express a new multi-partner contract that should be satisfied by the various parties involved. Fig. 4 presents the modified scenario.

2.3 Issues for business process changes

Providing support for adapting business processes raises several issues [1]. The first concerns correctness. Thus, after making a change to a business process, one should ensure that the resulting process is still correct and valid. Errors can be of two general kinds: independent of the application domain (what [1] call “syntactic errors”) or application-domain dependent (resp. “semantic errors”). For instance, the former may occur because the business process is ill-formed (with respect to the process language, e.g., WS-BPEL) or due to potential incorrect behaviour (such as deadlocks), whereas the latter may occur because the process does not obey anymore the business rules, e.g., changing the order of activities within a process so that clients have to pay before ordering. The second issue pertains to migrating business process instances, while they are running, to the new schema. Before migration, an instance that is in a specific state should be transferred, within the new schema, to an equivalent state. Therefore, equivalent states in the old vs. new business process schemas must be identified.

Finally, the third issue concerns how information about the current state of running business process instances should be presented for management purposes. This issue is complicated by the fact that many business process instances may be running at the same time.

3 Description of the CPC Protocol

Changes to a collaborative business process can have an impact on the contract specified between the parties involved. For instance, the changes on the business scenario described in Section 2 affect the buyer-seller contract. Therefore, each business process executed at each end-point may need to be adapted to meet the new contract. For this reason, we propose a new approach to support adaptability for collaborative business processes. This approach uses a new protocol, called Change Protocol for Collaboration (CPC), for managing the changes that can have incidences on the contract. In this section, we first present the protocol's key features. Then, we describe each protocol message. We conclude by discussing activities performed by trading partners at each protocol state.
3.1 Overview of the CPC protocol

The CPC protocol is a two-phase commit-like protocol and consists of five messages: Notify, Accept, Deny, Proceed and Cancel (see Fig. 5 and Fig. 6). When a trading partner (playing the Master role) wants to change its business process and this change can affect its partners (playing the Slave role), it sends a message notifying its partners of this change (1). Upon receiving this message, the slaves enter the Notified state (a slave is initially in state idle). The slaves can then accept (2) or deny (2') adapting their business processes. When a slave agrees to the change, it enters the Accepted state; otherwise, it enters the Denied state. For the master, two cases are then to be considered. If it receives an Accept message from all its slaves, it sends a Proceed message to inform its partners that they should adapt their business processes (3). Upon receiving this message, a slave enters the Proceed state. The second case is when the master receives a Deny message from a slave: in this case, it sends a Cancel message to all its slaves informing them that the change is canceled (3'). All slaves then enter the Canceled state. Note that canceling a change does not mean that a slave has a power to veto; it only means that this slave cannot adapt its business process in order to meet the new requirement. The master will then simply react by resubmitting the change after replacing the partners (slaves) that denied.

Fig. 7 shows a UML activity diagram describing the Master trading partner behavior, whereas Fig. 8 presents a UML statechart diagram for the Slave partner behavior. The activities performed by slaves (Get Authorization, Record Change Denied, Record Change Canceled and Program Migration) are described in Section 3.3. Note that at completion of one of the latter three activities, a slave trading partner returns to its initial state.

3.2 Description of the protocol messages
The Notify message consists of the following fields:
- **Message_ID**: Message identifier;
- **Effective_Date**: Date when the change will become effective;
- **Expiration_Date**: Date when business process will expire;
- **Scope**: Scope of the changes: Instance or Schema;
- **Running_Instances**: True or False = Whether the changes should be supported by running instances;
- **BusinessProcess_Schema**: Reference to the changed collaborative business process;
- **Instance_References**: Mandatory when the changes concern only some business process instances, in which case it indicates those instance references;

A Notify message would, for example, notify of the new contract expressed graphically in Fig. 4.

The Accept, Cancel and Proceed message consist of the following fields:
- **Message_ID**: Message identifier;
- **Notify_Message_RefID**: Corresponding Notify message identifier;

The Deny message consists of the following fields:
- **Message_ID**: Message identifier;
- **Notify_Message_RefID**: Corresponding Notify message identifier;
- **Reasons**: Text (free form) explaining why a notification change has been denied.

### 3.3 Activities of the Slave trading partners

The Get Authorization activity has two aspects. The first is purely business related, while the second is technical. First of all, the change should be authorized according to the organization policies. Then, a technical authorization should be obtained. The latter is obtained after verifying that the changed business process is correct (see Section 2.3). If so, the authorization activity should propose a set of modifications to be made to the local business process so that it satisfies the new contract. This local business process should be then validated and completed (manually) by the business analysts as well as the technical team. If a problem occurs at any stage, a Deny message is generated. Note that the verification process is considered beyond the scope of this paper.

The Program Migration activity consists in effectively migrating the running business process instances to the new schema, but only when the scope of changes is Instance or the **Running_Instances** field of the Notify message is True. When the scope of changes is Schema, the new schema is scheduled to become effective for all new business process instances based on the Notify message’s **Effective_Date** and **Expiration_Date** fields. This activity is detailed in the next section.

The Record activities (**Record Change Denied** and **Record Change Canceled**) consist only in recording those events for administration purposes.

### 4 Migrating running business processes

Migrating a running instance I1 of a business process from a schema S1 to a new schema S2 consists in four steps:
- Creating a new instance I2 of the business process according to the new schema S2;
- Extracting I1’s execution trace (let T be this trace);
- Compensating deleted **invoke** activities;
- Conditionally executing I2 according to S2 and T.

The first step is straightforward. The second one consists in extracting the trace of activities executed by I1, from its start activity until the last activity executed before it suspended. For instance, assume that invoking the Processing service to process the order is the last activity executed by an instance of the Seller’s business process according to the schema from Fig. 2. This instance’s execution trace then comprises the following flow of activities:
- Waiting for a **PurchaseOrderRequest** message (**receive** activity);
- Preparing a **CreditAuthRequest** message (**assign** activity);
- Sending a **CreditAuthRequest** message and receiving its associated **CreditAuthResponse** message (**invoke** activity);
- Selecting the **credit authorized case of the if**;
- Sending the **PurchaseOrderRequest** message to the Processing service (**invoke** activity).

The third step requires compensating, in reverse order of their execution, all **invoke** activities that do not appear in the new schema S2—as explained in the appendix, BPEL allows associating a compensate activity to an **invoke** activity. For our example, with the changes to the seller’s business process (see Fig. 3 and its corresponding BPEL description in the

---

1 More precisely, this trace reflects a BPEL description of Fig. 2 (see the appendix for a modified description of the new process expressed in Fig. 3) where a new activity (**assign**) is added in order to prepare sending the **CreditCheck** message to the financial institution.
appendix), there is nothing to compensate as all `invoke` activities are still present in the new schema.

The last step consists in performing a conditional execution of the new instance \( I_2 \) according to \( S_2 \) and \( \tau \). A conditional execution of a business process is executed as usual but with some modifications as expressed by rules R1, R2, R3 and R4.

**Rule 1 (R1):** The service invocation activity is skipped if the service type, its provider and the value of its input parameters did not change (i.e., already executed by \( I_1 \) and, therefore, belongs to \( \tau \)). However, a synchronization message is sent to the provider. This message will inform the provider (i.e., the partner) about the progress of the migrated instance on the requester’s side and, thus, will synchronize the different partners involved in the process. In fact, the synchronization message will simply contain the message to be sent to the partner.

**Rule 2 (R2):** Receiving a synchronization message has the same effect as receiving the message it contains. This means that a process can continue its execution if it is waiting for this message (for instance, the process is executing a receive activity).

**Rule 3 (R3):** The waiting activity of a period of time \( P_1 \) is executed only for a period of time \( P_1 - P_2 \). \( P_2 \) is equal to the time when the next activity \( B \) scheduled to be executed minus the time when the last activity \( A \) has been executed. The wait activity should be executed before activity \( B \). The idea here is to ensure the business process will not wait unnecessarily.

**Rule 4 (R4):** All `invoke` activities (not already compensated) that are executed by \( I_1 \) and cannot be reached by \( I_2 \) should be compensated in reverse order of their execution.

Returning to our example, migrating the instance to the new schema will proceed as follows. The newly created business process instance will start execution from the beginning according to the new schema. This process will wait for a synchronization message from the buyer (rule R2). The assign activity will be executed normally. Then the seller will send a synchronization message to the financial institution (rule R1). The instance will take the credit authorized branch like the old instance. It will then invoke the operation for checking the inventory. Depending on the reply from the inventory service, the new instance will select one of the back order or in stock branches. If the former is selected, invocation of the process order request’s activity (as well as other activities in the same branch) becomes unreachable. Rule R4 says that, in this case, this activity should be compensated as it has already been executed by the old instance. Finally, the new instance will continue its execution as usual.

## 5 Related Work

In this section, we describe how some commercial and academic BPMS provide support for adaptability. We also discuss approaches that propose solution for migration of running business process instances to their new schemas.

Microsoft BizTalk Server [14] offers two kinds of ports: static and dynamic ports. Static ports are used when the physical address of a service is known at design-time. BizTalk Server allows a user to manually change this port after deployment. Dynamic ports are used when the physical address is known only at run-time—this dynamic information is generally obtained from a message received from the provider of the service.

Oracle BPEL Process Manager [16] uses string substitutions applied to a configuration file where variables are used as properties. These variables can contain anything: operation names, server URLs, or other WS-BPEL attributes. It is then possible to assign dynamically a new address for a specific service by modifying the configuration file. Oracle BPMS also offers an interesting flexibility feature: Depending on user-specified rules, different actions can be taken. Thus, when the Process Manager meets a \(<\text{decide}>\) activity, it invokes its business rules engine to decide which action to select. These rules can be modified or augmented without redeployment.

IBM WebSphere Process Server [11] contains a business-rule component that provides support for rule sets ("if-then" rules) and decision tables. Selectors allow enhancing the behavior of business processes selectively based on business rules. A process can dynamically behave differently depending on business conditions.

Hewlett Packard eFlow [6] proposes different types of service nodes to support adaptability. First, it provides a dynamic service discovery based on a service selection rule specified at a service node. It also provides a multi-services node which, dynamically and repeatedly, can invoke multiple services of the same type. The number of service nodes to be activated is defined at run-time and can be determined, for instance, by the number of service providers for a given service. Another type of special node is the generic node, an abstract construct bound to various activities that are selected according to run-time characteristics. This allows dynamic process instance modifications to the process logic, to operations' definition, to routing conditions, or to data. These changes on process instances can be applied to a single instance or to a group of instances that meet some specified conditions.
Adams et al. [3] introduce the worklet concept in YAWL [21] to allow for the dynamic selection of actions for an activity. To these actions can be associated Web services of different types. When a process activity is identified as a worklet, its execution is performed by an external service that selects the appropriate action based on business rules. Business rules and actions can be modified or added even while the business process is executing.

Courbis and Finkelstein [7] use dynamic aspects [19] along with the visitor design pattern [9] to obtain a highly configurable and extensible WS-BPEL engine. Dynamic aspects are an extension of static aspects used in Aspect-Oriented Programming (AOP) [13]. Dynamic aspects can be woven or unwoven into/from a program “on-the-fly”. An aspect BPEL-specific language is proposed using XPath as a pointcut language to identify the join points (matching the WS-BPEL document) and WS-BPEL as the advice language. Using aspects for dynamic workflow adaptations has also been investigated in [4].

Karastoyanova et al. [12] emphasize the need for a unified way of expressing flexibility within process descriptions. They propose a meta-modeling approach consisting of two WS-BPEL meta-model extensions. The first is a new activity, called find_bind, which enables dynamic service discovery for a type of services. The second is another new activity, called evaluate, used when the types of services differ.

Sadiq et al. [17] propose the concept of pockets of flexibility within a process for supporting partially specified workflows. A pocket of flexibility consists of a set of workflow fragments and a special workflow activity called the build activity. The latter provides the rules for instantiating the pocket with a valid composition of fragments during the execution of a workflow instance.

The previous approaches are more suitable for partially specified business processes and do not need to support instance migration as changes are pre-planned. By themselves (without a complementary approach supporting instance migration), they have a serious limitation as planning the evolution of a business process is generally impossible. For instance, knowing exactly which Web services their providers could change is generally impossible. Of course, generalizing the use of dynamic ports is not a good solution as it has an impact on the performance of a BPMS.

Some did address the issue of instance migration within a workflow management system [6, 10, 18]. Basically, these approaches’ underlying ideas are similar. For instance, [6] identify two kinds of migration: unconditional and conditional. Unconditional migration occurs when the execution trace of an instance complies with the new schema—an instance complies with the new schema if it followed a path which is allowed by the new schema. Migration of an instance to a specific state consists in being transferred, within the new schema, to an equivalent state. Conditional migration requires some modification to the instance for it to comply with the new schema. These modifications consist in compensating some executed tasks. These approaches differ from ours in two ways. Firstly, they may unnecessarily compensate some activities. For instance, suppose that in the new schema, a new activity A3 is inserted between some activities A1 and A2. Suppose also that an instance already executed, within the old schema, A1 and A2. The migration will be possible only after compensating A2. In our approach, the compensation is required only if there is a change in the input parameters or provider. Secondly, they do not support instance migration in the context of distributed business processes.

6 Conclusion

In this paper, we proposed a new protocol, called Change Protocol for Collaboration (CPC), for managing the changes that can have incidences on the contract between business partners. This protocol ensures that all partners do adapt their process. We presented the key elements of the protocol. It is implemented using two Web services. The first service, called CPC Slave, has three operations: sendNotify, sendCancel and sendProceed. These operations’ input messages are respectively: Notify, Cancel and Proceed. The second one, called CPC Master, has two operations: sendAccept and sendDeny. Their input messages are respectively Accept and Deny.

We also proposed a new algorithm for supporting migration of running business process instances to their new schemas. Presently, we are implementing a business process engine that supports dynamic changes. To achieve this goal, we adopted the BPEL2Java (B2J) engine provided by the Eclipse Service Tools Project [8]. As future work, we also plan to tackle the challenge of verifying correctness of changes within the context of multi-partners business processes.

7 References


[2] The Accredited Standards Committee (ASC) X12. The
Appendix A

WS-BPEL description of the Seller process from Fig. 3

A process contains five major sections:

- `<partnerLinks>`: Define the different parties involved in the process. For our example (see below), five partner links have been defined; three links for the business partners involved in the business process and the remaining two for invoking internal Web services.
- `<variables>`: Define the data variables used by the process.
- `<correlationSets>`: Define groups of tokens used for message correlation, i.e., for routing messages to the correct process instance. For our example, the fields `customerID` and `orderNumber` are used to correlate messages exchanged between partners. Note that they are initialized when the seller receives a purchase order request message.
- `<faultHandlers>`: Specify the activities that must be performed in response to faults resulting from the invocation of Web services.
- `<Activity>`: Describe the process normal (dynamic) behavior.

BPEL provides two categories of activities: basic and structured activities. Basic activities correspond to atomic actions: `<invoke>`, invoking some Web service’s operation; `<receive>`, waiting for a matching message from a partner; `<reply>`, sending a message in reply to a message that was received by an inbound message activity; `<wait>`, waiting for a certain period of time or until a certain point in time has been reached; `<assign>`, updating the
value of a variable; <throw>, signaling a fault from inside the process; <rethrow>, rethrowing a fault originally caught by a fault handler; <compensate> and <compensateScope>, undoing the effects of already completed activities; <exit>, aborting the entire process instance; <validate>, validating the values of variables against their associated definition; <empty>, doing nothing (generally used for synchronization of concurrent activities).

Structured activities impose constraints on a set of activities contained within them. They include: <sequence>, for defining a sequential execution order; <flow>, for parallel execution; <if>, for conditional execution; <pick>, for describing a choice between one of possible messages to arrive or for a timeout to occur; <while>, <repeatUntil> and <forEach>, for looping; and <scope>, for grouping activities into blocks to which event, fault and compensation handlers may be attached. Note that a compensation activity may be associated with an invoke activity. For instance, SendOrderToShipper has an associated compensation activity.

```xml
<<?xml version="1.0" encoding="UTF-8"?>
<process poExample="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:poExample="http://www.latece.uqam.ca/CPC/BPELIEngine/POexample"
<!-- Here a set of import of WSDL message declarations -->
<!-- Here partnerlinkType declarations -->
<partnerLinks>
  <partnerLink name="Buyer" partnerLinkType="purchaseOrderLink" partnerRole="buyer"
    myRole="seller"/>
  <partnerLink name="FinancialInst" partnerLinkType="creditCheckLink"
    partnerRole="creditProvider"/>
  <partnerLink name="Shipper" partnerLinkType="shipperLink" partnerRole="shipper"/>
  <partnerLink name="Processing" partnerLinkType="processPOLink"
    partnerRole="processPOService"/>
  <partnerLink name="Inventory" partnerLinkType="inventoryLink"
    partnerRole="inventoryService"/>
</partnerLinks>
<!--Here variable declarations -->
<variables>
  <variable name="PORequest" messageType="poExample:PurchaseOrderRequest"/>
  <variable name="POResponse" messageType="poExample:PurchaseOrderResponse"/>
  <variable name="BOResponse" messageType="poExample:BackOrderResponse"/>
  <variable name="AuthRequest" messageType="poExample:CreditAuthRequest"/>
  <variable name="AuthResponse" messageType="poExample:CreditAuthResponse"/>
  <variable name="SORequest" messageType="poExample:ShippingOrderRequest"/>
  <variable name="SOResponse" messageType="poExample:ShippingOrderResponse"/>
  <variable name="InvResponse" messageType="poExample:InventoryResponse"/>
</variables>
<correlationSet name="PurchaseOrder" properties="poExample:customerID
  poExample:orderNumber"/>
<!--Start of process-->
<sequence>
  <receive partnerLink="Buyer" operation="sendPurchaseOrderRequest" variable="PORequest"
    createInstance="yes">
    <correlations>
      <correlation set="PurchaseOrder" initiate="yes"/>
    </correlations>
    </receive>
  <assign>
    <copy>
      <from variable="PORequest" part="customerInfo"/>
      <to variable="AuthRequest" part="customerInfo"/>
    </copy>
    <copy>
      <from variable="PORequest" part="totalPO"/>
      <to variable="AuthRequest" part="amount"/>
    </copy>
  </assign>
  <invoke partnerLink="FinancialInst" operation="sendCheckCredit"
    inputVariable="AuthRequest" outputVariable="AuthResponse"/>
</sequence>
```
<if><condition>$AuthResponse/authNumber = 0$</condition><!-- credit denied -->
<sequence><assign><copy><from variable="PORequest"/><to variable="POResponse" part="PO"/></copy><copy><from><literal>Credit authorization denied</literal></from><to variable="POResponse" part="comment"/></copy></assign><invoke partnerLink="Buyer" operation="sendPurchaseOrderResponse" variable="POResponse"/></sequence><else><!-- credit authorized -->
<sequence><invoke partnerLink="Inventory" operation="sendCheckInventory" inputVariable="PORequest" outputVariable="InvResponse"/><if><condition>$InvResponse/availability = 0$</condition><!-- back order -->
<sequence><assign><copy><from variable="InvResponse" part="BO"/><to variable="BOResponse" part="PO"/></copy></assign><invoke partnerLink="Buyer" operation="sendBackOrderResponse" variable="BOResponse"/></sequence><else><!-- in stock -->
<sequence><invoke partnerLink="Processing" operation="sendProcessOrderRequest" inputVariable="PORequest"> <compensationHandler><invoke partnerLink="Processing" operation="CancelProcessOrder" inputVariable="PORequest"/></invoke></compensationHandler></invoke><flow><sequence><assign><copy><from variable="PORequest"/><to variable="ShippingOrder" part="order"/></copy></assign><invoke partnerLink="Shipper" operation="sendOrderToShipper" inputVariable="ShippingOrderRequest" outputVariable="ShippingOrderResponse"> <compensationHandler><invoke partnerLink="Shipper" operation="CancelOrderToShipper" inputVariable="ShippingOrderResponse"/></invoke></compensationHandler></invoke><sequence><invoke partnerLink="Processing" operation="sendInvoiceRequest" inputVariable="PORequest"> <compensationHandler><invoke partnerLink="Processing" operation="CancelInvoiceRequest" inputVariable="PORequest"/></invoke></compensationHandler></invoke></sequence></flow></sequence></if>
<invoke operation="CancelInvoice"
        inputVariable="PORequest"
        />
</compensationHandler>
</invoke>
</flow>
<assign>
    <copy>
        <from variable="PORequest"/>
        <to variable="POResponse" part="PO"/>
    </copy>
    <copy>
        <from><literal>Tracking Number : $ShippingOrder/trackingNumber</literal>
        <from>
        <to variable="POResponse" part="comment"/>
    </copy>
</assign>
<invoke partnerLink="Buyer" operation="sendPurchaseOrderResponse"
        variable="POResponse"/>
</sequence>
</if>
</else>
</if>
</sequence>
</else>
</if>
</sequence>
</process>