

# WS-BPEL prototype for performing ebXML BPSS One-Action BusinessTransactions

Andreas Schönberger  
Distributed and Mobile Systems Group  
Otto-Friedrich-University of Bamberg  
Bamberg, Germany  
Email: andreas.schoenberger@uni-bamberg.de

## 1. Introduction

ebXML BPSS (ebBP) [1] is a dedicated B2Bi choreography standard that allows for the specification of B2Bi processes in a declarative and technology-agnostic way. ebBP's core concepts are so-called BusinessTransactions (BT) and BusinessCollaborations (BC). BTs are used to specify the exchange of up to two business documents. BCs then can be created by choreographing BTs. Due to its declarative nature, ebBP is well-suited as means for agreement between integration partners. Note that this is not an easy task as B2Bi projects necessitate the participation of personnel with widely varying skills and vocabulary (different enterprises, business analysts and software architects). ebBP does not specify technical details for performing BCs/BTs in order to allow for different implementation technologies, say Web services, AS2 [2] or ebMS [3], [4].

The prototype presented is implemented using Web services and WS-BPEL [5] technology. This is beneficial in terms of overcoming platform heterogeneity and wrapping legacy systems. Integration architecture plays an important role in the design of an ebBP BT execution prototype. In [6], a distributed integration architecture for performing ebBP choreographies has been proposed. Its core characteristics are modularization and separation of control flow logic from business logic. For each ebBP BT/BC a separate set of so-called control processes is applied for implementing control flow and for dealing with distributed computing issues. In order to handle legacy systems, business logic is assumed to be encapsulated by backend systems. The backend systems signal the need for new BT/BC executions to the control processes which in turn call back the backends' business document creation and validation facilities. The interaction between control processes and backends of an integration partner is assumed to be safe in the sense that messages do not get lost due to unreliable media or system crashes.

Figure 1 shows a modularized integration scenario of two integration partners A and B. At the BC level, there is a backend component as well as a control process for each

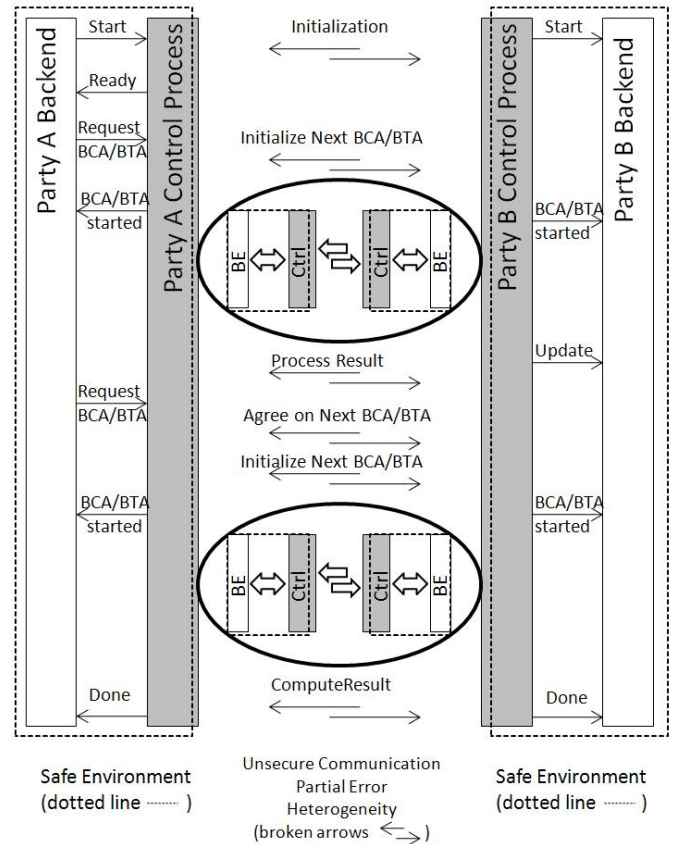


Figure 1. Modularized B2Bi integration scenario

integration partner (long vertical boxes). Partner A starts out with detecting the need for performing the agreed-upon BC. Accordingly, A's backend sends a *Start* message to A's control process which, in turn, initiates the BC together with B's control process. B's control process notifies B's backend that a new BC instance has been started. Subsequently, A's backend gets notified that the collaboration initialization has been finished and then requests the execution of a lower level BT. The control processes then negotiate BT parameters like an instance identifier or a time limit and pass on control to the lower level BT control processes (presented within the oval forms) which eventually produce a result value. This

result value is then used for routing the control flow. In this fashion, BTs defined in the ebBP choreography will be performed until an end state has been reached.

While this scenario leaves out several details about BC level control process interaction, it provides the context for the ebBP BT execution prototype. For more research work on performing B2Bi choreographies as WS-BPEL processes, see <http://www.uni-bamberg.de/pi/mitarbeiter/schoenberger/> or <http://www.big.tuwien.ac.at/staff/chuemer.html?area=publications>

## 2. Abstract Execution Model

The prototype implements an ebBP One-Action BusinessTransaction that is used for specifying the exchange of a single business document (therefore the name One-Action) between the so-called *requesting role* and *responding role*. ebBP's *DataExchange* business transaction type has been selected for modeling. According to ebBP ([1], page 35), this allows for free selection of ReceiptAcknowledgement (RA) and AcceptanceAcknowledgement (AA) business signals. The receiver of the business document can use RA and AA to give the sender of the business document information about the business document's state of processing. RA is used for signaling receipt or, if the *isIntelligibleCheck-Required* parameter is flagged, for signaling legibility of the document. AA is used for signaling that the business document "[...] has been accepted for business processing and that processing is complete and successful by the receiving application, service or a receiving business application proxy" ([1], section 3.4.9.3).

This BPEL prototype also contains the according ebBP model which is a model of RosettaNet's<sup>1</sup> PIP 3A20 that can be used for exchanging a Purchase Order Confirmation. The example ebBP model of PIP 3A20 is available as part of this prototype's artifacts.

The prototype's BPEL processes implement the control processes for performing a ebBP One-Action BT. The control flow is abstractly defined by means of state machines as visualized in figures 2 and 3. Message exchanges are modeled as `<communication role> {!,?}<message type>` where ! denotes an outgoing message while ? denotes an incoming message. The following communication roles are defined:

- REQ, the requestor's control process.
- RES, the responder's control process.
- MA, the superordinate (master) collaboration process triggering the execution of the BT. Note that each control process is triggered by its own MA process. Essentially there are two superordinate processes that have to coordinate with respect to starting a new BT execution.

1. <http://www.rosettanet.org>

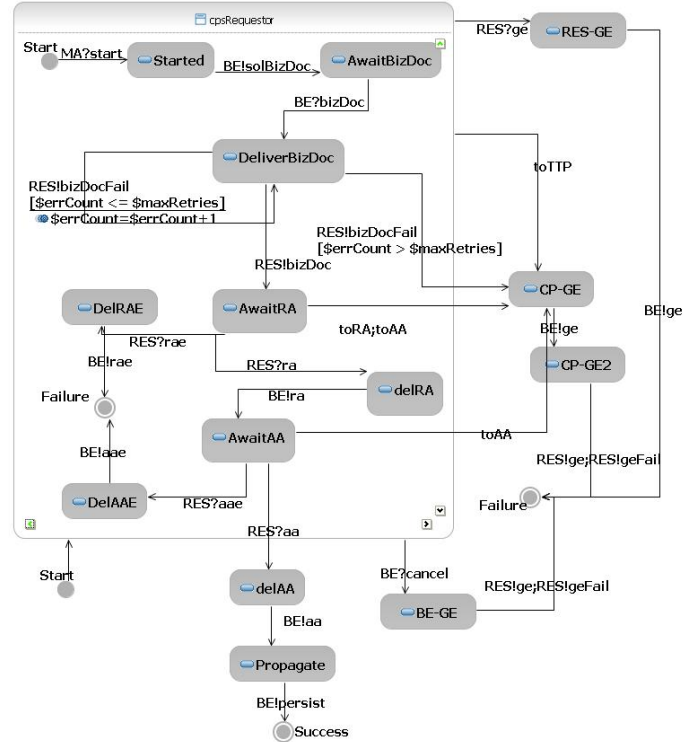


Figure 2. Requester Control Process Machine

- BE, the backend implementing business logic. Again, there are actually two backend implementations, one for each integration partner.
- RAC, the service for creating ReceiptAcknowledgements after having performed storage and legibility check procedures.

The following message types are defined:

- bizDoc, the business document to be exchanged.
- ra, the ReceiptAcknowledgement.
- aa, the AcceptanceAcknowledgement.
- rae, the ReceiptAcknowledgementException.
- aae, the AcceptanceAcknowledgementException.
- ge, ebBP's GeneralException that for all exceptional cases not covered by RAE and AAE.
- start, for triggering the BT execution.
- solBizDoc, for requesting the business document from the backend.
- cancel, for canceling the BT execution.
- persist, for signaling that the BT execution has been performed successfully from a protocol perspective.

Finally the following local event types are defined:

- toTTP, a timeout for ebBP's timeToPerform parameter.
- toAA, a timeout for ebBP's timeToAcknowledgeAcceptance parameter.
- toRA, a timeout for ebBP's timeToAcknowledgeReceipt parameter.
- <X>Fail, denoting the event that the sending process was not able to deliver a message of type X.

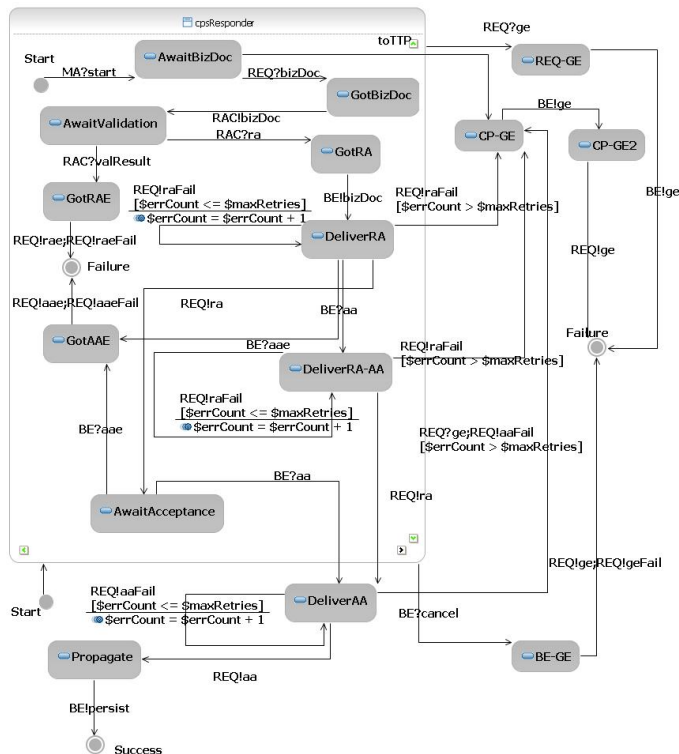


Figure 3. Responder Control Process Machine

The state-machines are performed pairwise which means, for example, that REQ communicates with RES, but it does not communicate with the responding party's BE. Message exchanges are performed synchronously which implies state machine transitions in lock-step.

### 3. Artifacts

This prototype contains the following artifacts:

- ebBP model:  
Contained in *artifacts/ebBP/ebbp-OneActionBT-basedOnPIP3A20.xml*  
Folder *artifacts/schema/* contains several relevant XML Schema and XML DTD definitions.
- BPEL processes:
  - *artifacts/BPEL/requestor* contains the requestor's BPEL definition together with all necessary WSDL and XSD definitions.
  - *artifacts/BPEL/responder* contains the responder's BPEL definition together with all necessary WSDL and XSD definitions.

### 4. Execution

This prototype's BPEL process definitions are ready for execution. The processes have been developed using OpenESB on top of GlassFish. If you intend to

execute the prototype, you may want to use the NetBeans6.7.1/GlassFishv.2.2+openESB bundle that has been used for developing the prototype (available free of charge at <https://open-esb.dev.java.net/Downloads.html>). The following description assumes that you are using a NetBeans/-GlassFish2.X bundle:

- 1) Create BPEL module projects for the *requestor* and *responder* control processes and copy the respective BPEL definitions into the created projects' ProcessFolders.
- 2) Create Composite applications for *requestor* and *responder* and add the BPEL modules.
- 3) For each Composite application, create test cases for each message consumed by BPEL processes except for the messages exchanged between *requestor* and *responder*. The relevant WSDL files for the *requestor* composite application are:

- PIP3A20RequestorBackendClient.wsdl
- PIP3A20RequestorMasterClient.wsdl

The relevant WSDL files for the *responder* composite application are:

- PIP3A20ResponderBackendClient.wsdl
- PIP3A20ResponderMasterClient.wsdl
- PIP3A20ResponderRACResult.wsdl

- 4) Create Web Service implementations for all outgoing messages of the BPEL processes (again except for the messages exchanged between *requestor* and *responder*). One way to do this is creating EJB modules and then using the prototype's WSDLs to create EJB implementation stubs. The relevant WSDL files for the *requestor* composite application are:

- PIP3A20RequestorBackendCallback.wsdl
- PIP3A20RequestorMasterCallback.wsdl

The relevant WSDL files for the *responder* composite application are:

- PIP3A20ResponderBackendCallback.wsdl
- PIP3A20ResponderMasterCallback.wsdl
- PIP3A20ResponderRAC.wsdl

- 5) Depending on your deployment setting (one host vs. different hosts; port setting), you may have to adjust the Web services' ports (due to BPEL, we're still sticking to WSDL 1.1).
- 6) Make sure that the test data used for correlation is assigned properly (can be taken from the WSDL definitions).
- 7) Deploy the composite applications and Web service implementations.
- 8) Perform the Web Service message exchanges according to the state machines.

As BPEL is a standard language you should be able to execute the process definitions on different BPEL engines as well. If you intend to do this, you will have to remove some GlassFish+OpenESB extensions used.

- GlassFish+OpenESB's {<http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace>}:trace tags have been used for creating logging statements. You can safely remove logging statements without modifying functionality.
- GlassFish+OpenESB's facility for creating user-defined exceptions has been used for creating exception specific faultHandlers. If such functionality is not available on your BPEL engine, then you have to remove the exception types scoped with the `procFaults` prefix and use a standard exception type. You then could use global exception type variables to distinguish different exception situations by assigning the variables immediately before throwing an exception and switching across the variables' values in a general faultHandler.
- GlassFish+OpenESB's {<http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/XPathFunctions>}:current-dateTime() function has been used for creating timestamps. If such functionality is not available on your BPEL engine, then you may want to use a dedicated Web service for this functionality.

## References

- [1] OASIS, *ebXML Business Process Specification Schema Technical Specification*, 2nd ed., OASIS, December 2006.
- [2] D. Moberg and R. Drummond, *MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)*, The Internet Engineering Task Force (IETF), July 2005. [Online]. Available: <http://www.ietf.org/rfc/rfc4130.txt>
- [3] OASIS, *ebXML Message Service Specification*, 2nd ed., OASIS, April 2002. [Online]. Available: [http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS\\_v2\\_0.pdf](http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf)
- [4] —, *ebXML Messaging Services Version 3.0: Part 1, Core Features*, OASIS, October 2007. [Online]. Available: [http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/ebms\\_core-3.0-spec-os.pdf](http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/ebms_core-3.0-spec-os.pdf)
- [5] —, *Web Services Business Process Execution Language*, 2nd ed., April 2007.
- [6] A. Schönberger and G. Wirtz, "Using variable communication technologies for realizing business collaborations," in *Proceedings of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), International Workshop on Services Computing for B2B (SC4B2B), Bangalore, India*. IEEE.