# QoS-Enabled Business-to-Business Integration Using ebBP to WS-BPEL Translations

Andreas Schönberger, Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb and Guido Wirtz
*Distributed and Mobile Systems Group*
*Otto-Friedrich-University of Bamberg*
*Bamberg, Germany*
{*andreas.schoenberger* | *guido.wirtz*}*@uni-bamberg.de*

## Abstract

*Business-To-Business Integration (B2Bi) is a key mechanism for enterprises to gain competitive advantage. However, developing B2Bi applications is far from trivial. Inter alia, agreement among integration partners about the business documents and the control flow of business document exchanges, applying suitable communication technologies for overcoming heterogeneous IT landscapes as well as ensuring a Quality of Service (QoS) level that is sufficient for B2Bi are major challenges.*
*In this context, applying choreography languages like ebXML BPSS (ebBP) for agreement among integration partners, orchestration languages like WS-BPEL for specifying partner-specific behavior, and Web Services for communication promises seamless interactions among business partners. In this scenario, the conformance of orchestration models to choreography models and cost-effective development are of paramount importance. Consequently, top-down approaches that automatically translate choreography models into orchestration models have been proposed.*
*By now, the realization of QoS attributes has not yet received the necessary attention that makes such approaches suitable for B2Bi. In this paper, we describe a proof-of-concept implementation of a translation of ebBP choreographies into WS-BPEL orchestrations that respects B2Bi relevant QoS attributes.*

**Keywords:** B2Bi, ebXML BPSS, WS-BPEL, Quality of Service, NES, conformance

## 1. Introduction

Supply Chain Management (SCM) is a key success factor for enterprises in today's business world [1]. Hence, Business-To-Business Integration (B2Bi) as a core SCM task deserves extensive treatment by industry and academia. The tasks to be solved in the B2Bi domain are challenging. Among others, the integration partners have to agree on the control flow of the integrated processes and on business document formats, ensure that the local processes implemented by each partner's IT systems conform to this agreement, and apply communication technologies that are suitable to interconnect their frequently heterogeneous IT facilities [2], [3], [4]. At the same time, B2Bi related Quality-of-Service (QoS) requirements like security and reliability have to be addressed. Today, Web services and the orchestration language WS-BPEL [5] (BPEL in the following) promise to provide the necessary tool set for exchanging business documents interoperably and for realizing the control flow between single Web service calls. Maturing Web service extensions like WS-Security [6] or WS-ReliableMessaging [7], frequently denoted as WS-* standards, can further be applied to realize QoS requirements. Yet, applying these technologies does not provide sufficient support for a global agreement upon control flow among integration partners. ebBP [8] as a choreography language is offering functionality for capturing the overall message flow between integration partners from a global point of view as well as for specifying B2Bi relevant QoS requirements. Hence, ebBP models may be used to assure such agreements. However, using ebBP choreographies as means for agreement and BPEL/Web Service based orchestrations for implementation rises the problem of conformance of orchestration models to choreography models. One obvious solution to this problem is creating a carefully designed translation engine that produces BPEL orchestrations from ebBP choreographies. Consequently, translating choreographies into orchestrations in general [9], [10], [11], [12], and ebBP to BPEL in particular [13], already has been proposed in literature. [9] even considers several QoS attributes but to the best of our knowledge a thorough treatment of QoS attributes as defined in ebBP during ebBP-to-BPEL translations has not yet been done.
This paper investigates this missing issue by modeling integration processes in ebBP and translating these models into QoS-enabled BPEL orchestrations. In order to ensure practical validity, real-world business processes from the Northern European Subset `www.nesubl.eu` (NES) have

been chosen as use cases. NES is a standardization body constituted of six northern European countries that define standard business document formats and standard business processes (profiles) in the area of electronic collaboration and procurement in both domestic and cross border trade.
The paper proceeds as follows: Section 2 gives a short overview of this work's most important technologies and section 3 introduces our approach to support QoS in B2Bi. Section 4 subsequently introduces the use case. The according ebBP models are then described in section 5 and their translation is outlined in section 6. Section 7 points out practical experience before related work is discussed in 8. Finally, section 9 concludes the paper and points out directions for future work.

## 2. Basics

ebBP, BPEL and QoS are core to our approach. The ebBP choreography language is based on the concept of *BusinessTransactions* (BT) that are used for exchanging one or two business documents between the so-called *RequestingRole* and *RespondingRole*. For the exchange of each business document, so-called *BusinessActivities* (BA) are used to specify whether *ReceiptAcknowledgements* and *AcceptanceAcknowledgements* as well as according exceptions are needed. So-called *BusinessCollaborations* (Coll) with at least two roles (integration partners) can be combined to build complex integration models. Usual control flow constructs like *Decision*, *Join* or *Fork* can be used to choreograph *BusinessTransactionActivities* (BTA) and *BusinessCollaborationActivities* (CA) that specify the actual execution of *BusinessTransactions* and *BusinessCollaborations*, respectively, by adding execution parameters such as *TimeToPerform* and mapping the roles of the performing *BusinessCollaboration* to the roles of the performed entity.
BPEL is a Web service orchestration language that is used for defining executable (or abstract) processes composed by a series of incoming or outgoing Web service calls. So-called *partnerLinkTypes* are defined within corresponding WSDL files that define *roles* in Web service communications based on WSDL *portTypes*. The BPEL process under consideration then uses *partnerLink* definitions for incorporating the *roles* of *partnerLinkTypes* and thus defines the functionality consumed or offered by the process. Based on these *partnerLinks* synchronous and asynchronous interactions like *invoke*, *receive* or *onMessage* can be defined and constructs like *sequence*, *if* and *while* can be used to define the control flow between these interactions. For details please refer to [5].
QoS are frequently defined as quantifiable, non-functional aspects of a service, but as security qualities like encryption or confidentiality are hard to quantify this work adopts the more general definition of [14]: *"[...]the term QoS [...is] used to denote all non-functional aspects of a service which may be used by clients to judge service quality. This extends other more restrictive QoS definitions such as the common interpretation of QoS to mean network performance attributes."*

## 3. Approach

We propose a top-down development approach that adopts the idea of automatically translating ebBP choreographies into BPEL orchestrations. In this scenario, ebBP is used for specifying the content of business documents and their order of exchange as well as according QoS properties. The

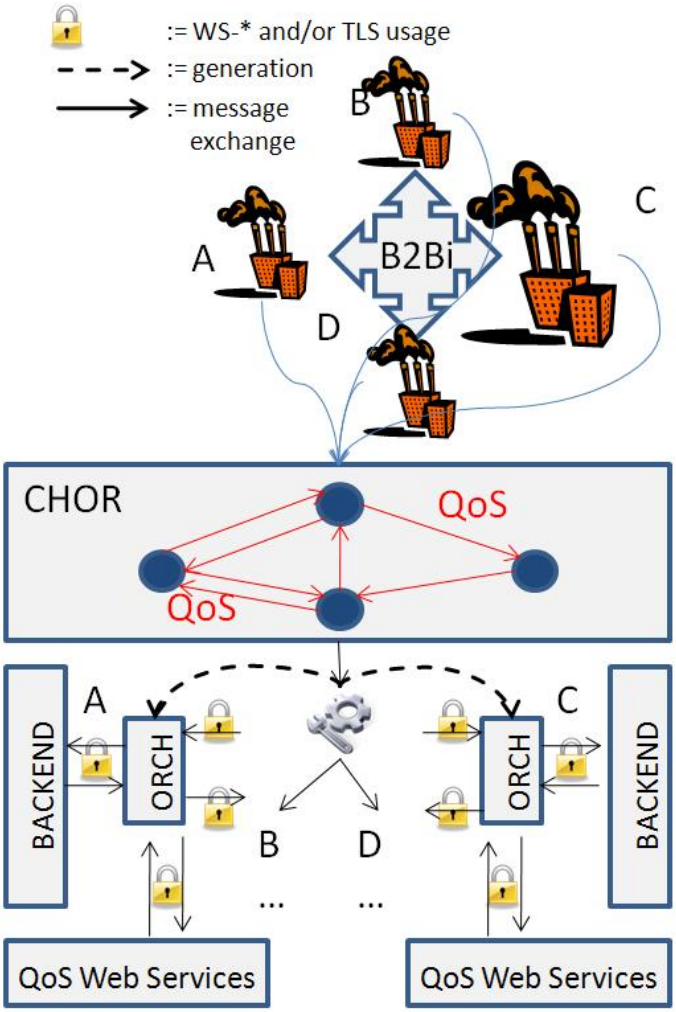


Figure 1. B2Bi integration architecture

BPEL processes to be generated are then responsible for implementing the protocol part of each integration participant using Web services as means for ensuring interoperability. We claim that an according ebBP-to-BPEL translator can ensure the conformance of these BPEL orchestrations to ebBP choreographies because the development of such a translator is a one-time effort. Clearly, a major issue in

writing an ebBP-to-BPEL translator is deriving a feature-complete mapping of ebBP constructs to BPEL constructs, but in this work we concentrate on the realization of B2Bi related QoS properties using Web service technology. We map a reduced set of ebBP constructs that we needed for modeling the NES use cases (cf. sec. 4) to BPEL. Despite these restrictions, our translation is still relevant to practice as the NES processes represent real-world B2Bi use cases.

Before discussing the realization of B2Bi QoS properties, the integration architecture our approach is based on is described. Figure 1 visualizes our architecture that assumes that real-world integration scenarios (upper third of the figure) are first captured in an choreography model defining QoS requirements (box denoted with CHOR) that is translated into according BPEL orchestration processes (denoted ORCH) afterwards, where each integration participant employs a BPEL process (control process in the following) on its own. These control processes get triggered by backend systems when a new B2Bi process instance is needed and then control the message flow of the collaboration by forwarding incoming messages to the backend systems and backend messages to the partners' control processes. Note, that business logic such as the creation and evaluation of business documents, e.g., *rating an order to be acceptable by creating a confirmation*, or capturing real-world events, e.g., *a new order has to be placed*, is encapsulated by backend systems using well-defined WSDL interfaces. Thus, the application of control processes helps in separating the message flow of a collaboration from the actual business logic. This idea is not new and has been proposed, among others, in the RosettaNet Implementation Framework [15] where so-called public and private processes are used to decouple the message exchange with integration partners from backend systems. This concept helps in wrapping existing legacy systems but also comes in handy in case no preexisting systems exist for parts of the needed business logic. Concerning our translator, the encapsulation of private logic assuming some well-defined interfaces also enables the derivation of BPEL control processes that do not have to be modified after generation. Within this scenario, Web services are proposed for both, the communication between partners and the communication between control processes and backend systems. We argue that interoperability and decoupling of systems is equally important for both types of communication.

The notion of QoS as defined in sec. 2 needs an operationalization for B2Bi. In order to identify the set of QoS requirements at least necessary for B2Bi, this work adopts the proposal of the B2Bi experts who authored the ebBP specification: *"The ebBP technical specification provides parameters that can be used to specify certain levels of security and reliability. This specification provides these parameters in general business terms"* ([8], sec. 3.5.7). Therefore, support for the QoS attributes identified in ebBP should be sufficient for B2Bi. These can be classified in attributes that relate to business documents and attachments (DocAtt), *BusinessActivitys* (BA), *BusinessTransactions* (BT), *BusinessTransactionActivitys* (BTA) and *BusinessCollaborations* (Coll, cf. sec. 2). Table 1 lists the QoS attributes identified in ebBP and the according level of specification. Most of these QoS attributes are self-explanatory and don't accept any parameters. Exceptions are the *DocAtt* attributes which are all related to security and the *timeToPerform* attribute. For the *DocAtt* attributes, ebBP distinguishes between realization options, i.e., the respective attribute has to be realized using security mechanisms at the transport level (*transient*), at the application level (*persistent*) or both (*transient-and-persistent*). Finally, *timeToPerform* offers parameters for specifying whether timers shall be set statically (*design*), be agreed upon by partners before executing collaborations (*configuration*) or during collaborations (*runtime*).

| QoS Attribute | Level of Specification |
|---|---|
| isAuthenticated | DocAtt |
| isConfidential | DocAtt |
| isTamperDetectable | DocAtt |
| isIntelligibleCheckRequired | BA |
| isNonRepudiationRequired | BA |
| isNonRepudiationReceiptRequired | BA |
| timeToAcknowledgeReceipt | BA |
| timeToAcknowledgeAcceptance | BA |
| isAuthorizationRequired | BA |
| retryCount | BA |
| isGuaranteedDeliveryRequired | BT |
| hasLegalIntent | BTA |
| isConcurrent | BTA |
| timeToPerform | Coll/BTA |

Table 1. ebBP QoS attributes and specification levels

Huemer and Kim [13] identified three groups of realization options for the QoS attributes described above: *"The first group uses a mapping to a corresponding BPEL attribute. The second group is reflected in the process structure. Finally, the third group does not have a corresponding BPEL concept. They must be handled by other standards of the Web Services stack."* This work adopts this analysis and adds the fourth option of home-grown QoS Web services which are then called by the BPEL orchestration processes for fulfilling a particular QoS task, e.g., performing an authorization check. Compared to the realization options in [13], these QoS services constitute a mixture of *reflecting the services in the process structure* and *other standards of the Web Services stack*. Further, according to the ebBP specification some attributes like encryption may be specified to be addressed on the transport level. Therefore, the integration architecture depicted in figure 1 also includes a visualization of QoS Web services which are called by the orchestration processes as well as padlocks representing the use of WS-* standards or

transport level security (TLS/SSL). A detailed discussion on QoS realization follows in section 6.

## 4. Use Case

Real world processes, namely NES profiles, are employed for investigating ebBP QoS attributes during ebBP to BPEL translations. The focus of NES is the definition of a subset of the Universal Business Language (UBL) [16] as a common business document format. The exchange of these documents (messages) is defined in eight so-called *profiles* that specify the exchange of one (profiles 3 and 4) up to 5 (profile 7) business documents. This is done by describing, among others, business benefits and requirements of the profile, usage scenarios, and a process specification using an UML activity diagram. For the purpose of demonstration, NES profile 1 is discussed in this and the subsequent sections. The remaining NES profiles are available from [17]. Figure 2 depicts the activity diagram of profile 1 which covers the exchange of a catalogue and its approval. The *supplier* role

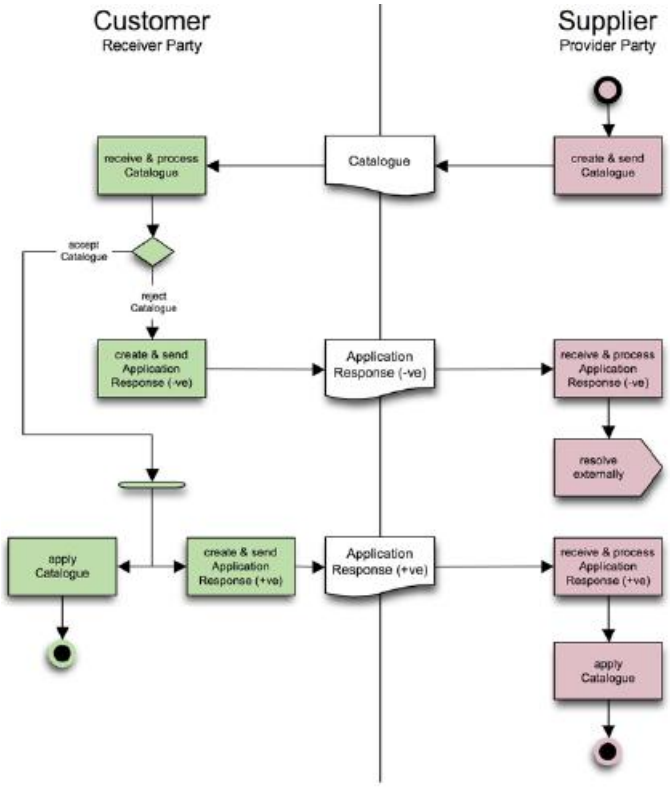


Figure 2. Activity diagram of NES profile 1[17]

of profile 1 sends a catalogue document to the *customer* role which then accepts or rejects the catalogue and subsequently sends back a positive or negative application response to the *supplier*.

During our investigation, all NES profiles have been modeled in ebBP and automatically translated into BPEL. The resulting BPEL processes in turn all have been XML schema validated and successfully deployed on the GlassFish[1]/openESB[2] BPEL engine. Moreover, for profiles 1-4 a dummy backend for triggering processes and creating/judging upon business documents has been implemented. Using these backends the actual flow of business documents and the realization of QoS properties have been tested.

## 5. ebBP Model

As NES does not provide a concise meta model for the profile business processes (it does so for the business documents), modeling the NES profiles using the ebBP constructs defined in section 2 is not based on an algorithm. Instead, the processes have been remodeled using the profile documentations and UML diagrams. As opposed to NES, ebBP models only capture the globally visible message flow of the choreography. For example, the NES model in figure 2 describes the decision process of the *Customer* role for the handling of the *catalogue* business document. It is irrelevant for the choreography whether the *Customer* applies the *Catalogue* and sends an *Application Response* in parallel or in sequence. Therefore, the ebBP model only captures the

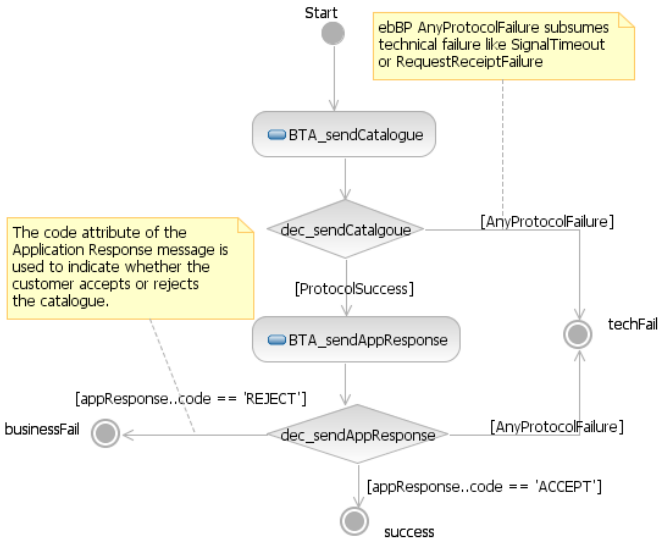


Figure 3. Visualization of the ebBP model of profile 1

information that is unconditionally needed for representing the global control and message flow. This is visualized in figure 3 that depicts an informal UML activity diagram of the ebBP model of profile 1. The ebBP model starts out with a *BusinessTransactionActivity* (BTA) for sending the catalogue from the *Supplier* to the *Customer* (these roles are specified in the textual ebBP representation) and then arranges for a check for technical success of the message transmission. Therefor, an ebBP *Decision* element is used that employs the

1. https://glassfish.dev.java.net/
2. https://open-esb.dev.java.net/

predefined ebBP condition guard values *ProtocolSuccess* and *AnyProtocolFailure* accordingly. Subsequently, another BTA is used to transmit the application response of the *Customer* and the *Customer's* decision is captured by the *code* attribute of the business message. Again, an ebBP *Decision* is used to model the different results of the message transmission. As can be seen, all information needed for describing the global message exchange is included in the ebBP model while aspects of the partners' local decisions, as contained in the NES models, are not described. Thus the ebBP model is better suited as basis for agreement among the integration partners whereas the NES model is better suited for describing the context of the business collaboration.
The assignment of ebBP QoS attributes is derived from the NES models by investigating the *intent* of each profile. For the purpose of investigating QoS in the ebBP to BPEL translation process, all available QoS attributes are activated for some profiles in order to ensure that all these can indeed be realized if needed. Listing 1 shows a *BusinessActivity* that is used for transmitting an order document. More QoS attributes are specified at the further ebBP elements as described in table 1.

Listing 1. QoS assignment of a BusinessActivity

```
<!-- Enclosed within a BusinessTransaction
     definition -->

<RequestingBusinessActivity
 name="sendOrder"
 nameID="bt_issueOrder_ba_req"
 isAuthorizationRequired="true"
 isIntelligibleCheckRequired="true"
 isNonRepudiationRequired="true"
 isNonRepudiationReceiptRequired="true"
 retryCount="3"
 timeToAcknowledgeReceipt="PT1H"
 timeToAcknowledgeAcceptance="P1D">
 <!-- more subtags defining
   documentEnvelopes and acknowledgements
       -->
</RequestingBusinessActivity>
```

## 6. Translation Concept

The discussion of translating ebBP specifications into BPEL comprises the mapping of control flow constructs and the realization of QoS. For the work at hand, the minimum requirement for translating ebBP concepts into BPEL is being able to translate real-world processes. As a first step those ebBP constructs used for modeling the standardized NES profiles are translated whereas few other ebBP elements, in particular *Joins* and *Forks* which precludes parallel standard models, are not considered for translation.

Tables 2 and 3 show the mapping of the control flow part of our ebBP-to-BPEL translator. Basically, binary ebBP collaborations are translated into one parent BPEL control process per participant. The generated BPEL processes have a tree-like structure composed of the mapping code for *BTAs* and *Decisions* where the generated code for follow-on *BTAs* is placed within BPEL `if/else` elements that are used for mapping *Decisions*. Iterative execution of *BTAs* is therefore to be modeled by means of recursive invocations using *CollaborationActivitys*. The eBP *Start* element is used to determine which *BTA* to map first and has no direct BPEL code mapping. The ebBP *Failure* and *Success* elements are translated as BPEL `invokes` from the control processes towards the backends for signaling process termination. These calls represent the leaves of our generated process structure. The Web service calls used to transmit business documents are content agnostic, i.e., business documents are

| Mapping of *BusinessCollaboration* | |
|---|---|
| Role | Mapping Elements |
| Initiator | - `process` element as collaboration root process<br>- use *Role* for association of Initiator/Responder roles in according *BTAs*/*CAs*<br>- `receive` for receiving initiating signal from backend<br>- `invoke` on helper Wsrv for creating unique process id<br>- `invoke` on partner process to distribute process id<br>- `invoke` on backend to distribute process id<br>- `invoke` to signal backend that first bizDoc can be sent<br>- `receive` to wait for first bizDoc<br>- `onAlarm` to capture *TimeToPerform* value; in case TimeToPerform is of type *runtime* use several negotiation calls<br>- `faultHandlers` in case of business protocol failures and according `invokes` for informing partner processes and child processes (if any)<br>- `onEvent` for receiving protocol failure signals from partner process<br>- `sequence` for the first *BTA* and its *Decision* with content as defined in the according mappings |
| Mapping of *BusinessTransactionActivity* | |
| Role | Mapping Elements |
| Agnostic | - used to correlate *BusinessCollaboration Roles* with *RequestingRole* / *RespondingRole* of *BusinessTransaction* according to *Performs* tags<br>- `invoke` and `receive` to request next bizDoc from backend if not done within *BusinessCollaboration* scope<br>- `onAlarm` to capture *TimeToPerform* value; in case TimeToPerform is of type *runtime* use several negotiation calls |
| Mapping of *BusinessTransaction* | |
| Role | Mapping Elements |
| Request-ingRole | - `while` for *retryCount* of *RequestingBusinessActivity*<br>- `faultHandlers` to handle *ReceiptAcknowledgement -Exception* and continue with next while iteration<br>- `invoke` on helper Wsrv to create signature for bizDoc received in surrounding *BTA* / *Coll* code<br>- `invoke` to send bizDoc to partner process<br>- `flow` for handling *ReceiptAcknowledgement*/ *AcceptanceAcknowledgement* in a separate `scope` each. Therefor, `receive` to receive ebBP business signal and `onEvent` to receive ebBP business signal exceptions. Further, calls to signature check and archival helper Wsrv where necessary. Finally, `onAlarm` for capturing business signal timeouts and `throws` for signaling processing errors.<br>- *RespondingBusinessActivity* mapped symmetrically |

Table 2. ebBP to BPEL mapping I

| Mapping of *Decision* | |
|---|---|
| Role | Mapping Elements |
| Agnostic | - *ConditionGuardValue AnyProtocolFailure* mapped by `onAlarms`/`faultHandlers` of enclosing `scopes`<br>- Evaluation based on latest bizDoc exchanged.<br>- *FromLink* used for determining where to insert the *Decision* mapping code.<br>- Nested `if` / `else` structure where each *ToLink* is enclosed within `else` block of preceding *ToLink* code.<br>- XPath evaluation `invoke` of helper Wsrv if necessary.<br>- Include code for follow-on *BTAs* in branches of `if` / `else` structure |
| Mapping of *CollaborationActivity* | |
| Role | Mapping Elements |
| Agnostic | - used to correlate parent collaboration *Roles* with *inner* collaboration *Roles*<br>- new `process` for *inner* collaborations with `receive` to be created from parent process.<br>- adapted `onEvents` / `faultHandlers` for parent-child interaction |

Table 3. ebBP to BPEL mapping II

wrapped within a message container using the XML Schema *any* type. This message container carries several meta data like correlation information and a string that specifies the *messageType* under transmission which represents the ebBP *BusinessDocument* type definition. Accordingly, the ebBP *BusinessDocument* type is not reflected in the WSDL interfaces that are created for specifying control process or backend functionality.

The mapping of some ebBP elements, in particular *BusinessCollaboration* and *BusinessTransaction*, is role dependent, i.e., the selection and sequence of BPEL elements used to map the ebBP constructs varies depending on whether the process under consideration represents the integration partner who sends the first business document of the *BusinessCollaboration* or the first business document of a *BusinessTransaction*. Tables 2 and 3 only contain the mapping for one role of these constructs because the mapping code of the corresponding role is more or less symmetric. Moreover, `variable` handling as well as the details of nesting `scopes` and `sequences` are not described. Nonetheless, note that the control processes request the business documents to be exchanged from the backends outside the BPEL `scope` that is used to implement the exchange with the partner process in order to avoid delays in backend interaction being counted for within the BPEL `onAlarms` for the actual partner interaction. Finally note that *business document* and *Web service* are abbreviated as *bizDoc* and *Wsrv* respectively and that ebBP constructs are typeset like *ebBP* and BPEL elements like `BPEL`.

There are several possibilities for realizing QoS properties (cf. sec. 3). The realization option *BPEL elements* is chosen for several properties related to timeouts and for the ebBP *retry* construct. BPEL `onAlarms` are used for realizing timeouts where triggering `onAlarms` may require special treatment with respect to backend interaction (see above). Opposed to that, the treatment of *retry* with a BPEL `while` is straightforward. The second realization option *QoS Web services* (WSrv) is chosen for creating/validating XML signatures using the standard Java API, for performing XML and Schematron validations of business documents, for archival of messages, and for authorization checks against a simple user list. In particular, XML signatures are used for multiple purposes, namely for authentication (isAuthenticated attribute) and integrity (isTamperDetectable) reasons as well as for realizing non-repudiation. Note that, for the implementation of non-repudiation, signatures are not only checked upon arrival of a message but are also archived together with the corresponding message. All QoS Web Services are called by the control processes where necessary. Next, the WS-* standards WS-Security (isConfidential) and WS-ReliableMessaging (isGuaranteedDeliveryRequired) as well as TLS (document related attributes, cf. sec. 3) are used for realizing QoS properties. If document related attributes of type *transient* (see sec. 3) are specified, TLS as a network layer technology is used. Otherwise, if the type is *persistent*, WS-Security or QoS Web services are employed. WS-Policy [18] can be used to instruct the BPEL execution engine/Web service stack to apply WS-* standards as well as TLS to the communication. Therefor, WS-Policy assertions are to be included in the WSDL descriptions of the BPEL processes and Web services. As an example, a policy assertion activating the use of WS-ReliableMessaging is given in listing 2.

Listing 2. WS-Policy fragment for activating WS-ReliableMessaging

```
...
<wsp:Policy wsu:Id="
    BPEL2BPELNoQosRMServicePortBindingPolicy
    ">
 <wsp:ExactlyOne>
  <wsp:All>
   <wsaws:UsingAddressing
    xmlns:wsaws="http://www.w3.org/2006/05/
        addressing/wsdl" />
   <wsrm:RMAssertion />
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
...
```

Note that, depending on ebBP QoS attribute values, WS-Policy assertions for any combination of WS-Security, WS-ReliableMessaging and TLS are supported for communication. Finally, the ebBP QoS attributes *isConcurrent* and *hasLegalIntent* are not realized in terms of the realization options pointed out in section 3. According to ebBP, *isConcurrent* (cf. [8], section 3.4.10.1) specifies whether multiple instances of a *BusinessTransaction* within the same process or in different processes (with the same party) are allowed to be active at the same time. Regarding the generated BPEL processes, concurrent executions are no problem and we argue that a backend system that offers its functionality via a Web Service should be able to handle concurrent access. Therefore, *isConcurrent* is always set to true in this work.

The semantics of *hasLegalIntent* is not completely clear. ebBP states that *"The hasLegalIntent attribute could have widely differing interpretations and enforceability depending on type of business, process, and jurisdiction."* ([8], section 3.4.9.7). We propose that a user shall configure a mapping of *hasLegalIntent* to the assignment of other ebBP QoS attributes during the translation process. Table 4 summarizes the realization of ebBP QoS attributes.

| QoS Attribute | Realization strategy |
|---|---|
| isAuthenticated | transient: TLS<br>persistent: WSrv SignatureValidation |
| isConfidential | transient: TLS<br>persistent: WS-Security |
| isTamperDetectable | transient: TLS<br>persistent: WSrv SignatureValidation |
| isIntelligibleCheckRequired | WSrv XMLValidation<br>WSrv SchematronValidation |
| isNonRepudiationRequired | WSrv Archive<br>WSrv SignatureCreation |
| isNonRepudiationReceipt-Required | WSrv Archive<br>WSrv SignatureCreation |
| timeToAcknowledgeReceipt | BPEL `onAlarm` |
| timeToAcknowledgeAcceptance | BPEL `onAlarm` |
| isAuthorizationRequired | WSrv AuthorizationCheck |
| retryCount | BPEL `while` |
| isGuaranteedDeliveryRequired | WS-ReliableMessaging |
| hasLegalIntent | defined in terms of other QoS attributes |
| isConcurrent | not relevant |
| timeToPerform | BPEL `onAlarm` |

Table 4. ebBP QoS attributes and and realization strategies

## 7. Practical Experience

The ebBP to BPEL translator is written in the Java language and the main API used for that is the Java API for XML Binding (JAXB[3]). The translator comprises approximately 6800 method lines of code excluding JAXB mapping classes, implementation of QoS Web Services and backend implementation. For the generation of WSDL interfaces, the Apache Velocity template engine is used. GlassFish V2 and its Web service stack Metro together with the openESB BPEL engine as well as Apache Tomcat 6 together with Axis2 and ODE 1.2 have been evaluated as execution platform. The GlassFish combination has been chosen due to better QoS support. Though it was possible to offer BPEL processes as secure and reliable Web Services with Axis2, invoking other services from BPEL processes reliably and securely was not possible.
All NES profiles can be completely translated, fully BPEL compliant process descriptions are produced and these processes can successfully be deployed on the GlassFish execution platform. Further, for profile 1-4 dummy backend systems have been created for testing the generated BPEL processes. As expected, testing revealed several errors although the according BPEL processes deployed without any problems. Figure 4 shows the dummy interface for incoming messages which is used to acknowledge or decide upon system messages and business documents. In order to check the existence of encryption and signature tags on the network level, the network protocol analyzer WireShark[4] has been used. In so far, not having tested profiles 5-8 using such dummy backends limits the results of the work at hand to some extent. Nonetheless, supporting B2Bi relevant QoS attributes during ebBP to BPEL translations is possible.
A limitation of the openESB BPEL engine is that access to SOAP headers is not possible. Therefore, XML signatures had to be created using a utility QoS Web service although WS-Security offers the possibility for automatically generating signatures. Access to signatures is necessary as non-repudiation requires archival storage of signatures.

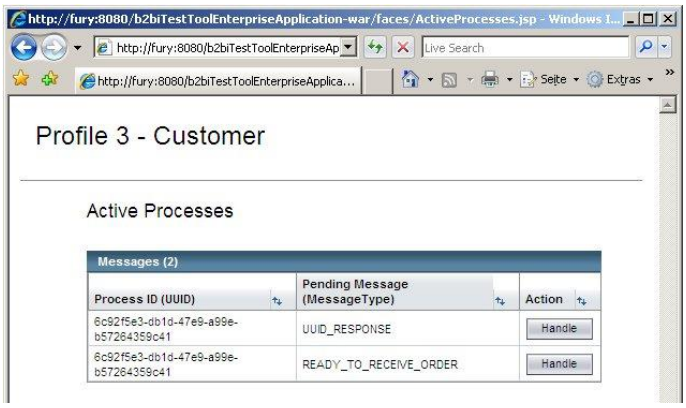


Figure 4. Backend interface for incoming messages

## 8. Related Work

There are various works that are similar to our approach with respect to translating choreographies into orchestrations. The majority of these does not describe a proof-of-concept implementation of a working translator ([13]), does only consider *BusinessTransactions* and not *BusinessCollaborations* ([19]), does not cover B2Bi QoS properties ([10]), and/or derives BPEL from WS-CDL and not ebBP ([12], [11]). As far as we know full support for ebBP QoS attributes during translation to BPEL has not yet been described.
Closest to our approach in terms of QoS support is [9] but, there, a completely different set of QoS attributes which are performance metrics related is used and the processes in their work have to be extended by *"private business logic"*. We are following a different approach that assumes backend

3. https://jaxb.dev.java.net/

4. http://www.wireshark.org/

system interfaces so that the BPEL processes do not have to be edited after generation.
With respect to using WS-CDL instead of ebBP as choreography language we claim that WS-CDL may be a good choice for many choreographies due to its tight relationship with WSDL as opposed to ebBP, but ebBP is particularly useful for B2Bi due to its better support for specifying QoS features (though non-standardized extensions are available [9]). Also, the embedding of ebBP in the ebXML framework seems to be advantageous for B2Bi scenarios.
Finally, it should be noted that integration standards organizations frequently do specify how to perform *BusinessTransactions* or similar concepts at runtime, e.g., RosettaNet specifies the so-called RosettaNet Implementation Framework which defines how to perform RosettaNet Partner Interface Processes. But these standards typically do not offer the generation of executable implementations of control processes as we do with our translator.

## 9. Conclusion and Future Work

In conclusion, this paper shows that support for the most important B2Bi QoS properties during translating ebBP choreographies into BPEL orchestrations is possible. An integration architecture for executing these orchestrations as well as a working proof-of-concept implementation of an ebBP to BPEL translator have been described. The work was evaluated using real world B2Bi processes, namely the NES profiles.
Currently, only the ebBP modeling elements needed for modeling NES profiles can be translated. Future work therefore concerns full support for all ebBP modeling elements, in particular multi-party processes and parallel execution using forks and joins. Further, extending the support for more technical details of QoS properties, e.g., the encryption algorithms to use, by using functionalities of ebXML Collaboration Protocol Agreement (CPA) is planned. Finally, our ebBP-to-BPEL translator currently assumes a top-down development model which is also due to the BPEL process structures created. Therefore, a modular structure that lends itself better to bottom-up or meet-in-the-middle approaches is planned.

## References


[1] D. M. Lambert and M. C. Cooper, "Issues in supply chain management," *Industrial Marketing Management*, vol. 29, no. 1, pp. 65 – 83, 2000.

[2] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K.Elmagarmid, "Business-to-business interactions: issues and enabling technologies," *The VLDB Journal*, vol. 12, no. 1, pp. 59–85, 2003.

[3] C. Schroth, T. Janner, and V. Hoyer, "Strategies for cross-organizational service composition," in *MCETECH '08: Proc. of the 2008 Intern. MCETECH Conference on e-Technologies*. Washington, DC, USA: IEEE CS, 2008, pp. 93–103.

[4] A. Schönberger and G. Wirtz, "Taxonomy on consistency requirements in the business process integration context," in *2008 Conf. on Software Engineering and Knowledge Engineering (SEKE'2008)*, Redwood City, USA, July 2008.

[5] OASIS, *Web Services Business Process Execution Language*, 2nd ed., April 2007.

[6] ——, "Web Services Security v1.1," February 2006.

[7] OASIS, *Web Services Reliable Messaging (WS-1 ReliableMessaging) Version 1.1*, OASIS, January 2008.

[8] OASIS, *ebXML Business Process Specification Schema Technical Specification*, 2nd ed., OASIS, December 2006.

[9] F. Rosenberg, C. Enzi, A. Michlmayr, C. Platzer, and S. Dustdar, "Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL," in *EDOC'07: Proc. of the 11th IEEE Intern. Enterprise Distributed Object Comp. Conf.*, Washington DC, USA, 2007.

[10] B. Hofreiter and C. Huemer, "A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL," in *Joint Conf. CEC'08 EEE'008*. Washington DC, USA: IEEE, 7 2008.

[11] J. Mendling and M. Hafner, "From WS-CDL choreography to BPEL process orchestration," *Journal of Enterprise Information Management (JEIM).*, vol. 21, no. 5, 2008.

[12] I. Weber, J. Haller, and J. A. Mülle, "Automated derivation of executable business processes from choreographies in virtual organizations," in *In: Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Band 2, XML4BPM Track, GITO-Verlag Berlin*, Mar. 2006, pp. 313–327.

[13] J.-H. Kim and C. Huemer, "From an ebXML BPSS choreography to a BPEL-based implementation," *SIGecom Exch.*, vol. 5, no. 2, pp. 1–11, 2004.

[14] G. Dobson, R. Lock, and I. Sommerville, "QoSOnt: a QoS ontology for service-centric systems," in *EUROMICRO'05: Proc. of the 31st EUROMICRO Conf. on Software Eng. and Advanced Applications*. Washington DC, USA: IEEE, 2005.

[15] *RosettaNet Implementation Framework: Core Specification*, V02.00.01 ed., RosettaNet, www.rosettanet.org, March 2002.

[16] J. Bosak, T. McGrath, and G. K. Holman, *Universal Business Language v2.0*, OASIS, December 2006.

[17] *Northern European Subset Profiles*, 2nd ed., Northern European working group for development of a subset for UBL 2.0, July 2007. [Online]. Available: http://www.nesubl.eu

[18] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ümit Yalçinalp, *Web Services Policy 1.5 - Framework*, W3C, September 2007.

[19] B. Hofreiter, C. Huemer, P. Liegl, R. Schuster, and M. Zapletal, "Deriving executable BPEL from UMM business transactions," in *IEEE SCC*. IEEE CS, 2007, pp. 178–186.