A Composable, QoS-aware and Web Services-based Execution Model for ebXML BPSS BusinessTransactions

Andreas Schönberger and Guido Wirtz
Distributed and Mobile Systems Group
Otto-Friedrich-University of Bamberg
Bamberg, Germany
andreas.schoenberger@uni-bamberg.de
guido.wirtz@uni-bamberg.de

Christian Huemer
Business Informatics Group
Vienna University of Technology
Vienna, Austria
huemer@big.tuwien.ac.at

Marco Zapletal
Electronic Commerce Group
Vienna University of Technology
Vienna, Austria
marco@ec.tuwien.ac.at

Abstract

Adequate IT support for Business-to-Business integration (B2Bi) is indispensable in today's globalized world. Agreement among personnel from different enterprises as well as distributed computing issues are major challenges to the automation of B2Bi processes. These challenges can be addressed by applying the choreography language ebXML BPSS (ebBP) for declaratively specifying B2Bi processes and using Web services and WS-BPEL as dedicated integration technologies. ebBP BusinessTransactions (BT) are the primary building block of ebBP choreographies and specify the exchange of up to two business documents in a declarative and technology-agnostic way. Composing BTs within choreographies and realization of QoS raise important requirements for the orchestration layer. This paper investigates these requirements and presents a composable, abstract, flexible and QoS-aware execution model that can be implemented using Web Services and BPEL.

Keywords: B2Bi, choreography, orchestration, ebXML BPSS, WS-BPEL

1. Introduction

The importance of B2Bi is evidenced by the work of numerous B2Bi standardization bodies. For example, RosettaNet[1] reports that its members "*transact billions of dollars in transactions within their trading networks using RosettaNet Partner Interface Processes (PIPs)*". Yet, B2Bi is far from trivial as personnel from different enterprises with different vocabulary and background have to agree upon the business documents to exchange, control flow of document exchanges and the implications of document exchanges. *Atomic* building blocks have been identified as facilitator for this agreement task ([1], [2]). ebXML BPSS (ebBP) [3] is a dedicated B2Bi choreography language that offers so-called BusinessTransactions (BT) as atomic building blocks. BTs specify the exchange of up to two business documents together with Quality-of-Service (QoS) and state alignment parameters. For B2Bi processes of more realistic size, BTs can be composed within so-called BusinessCollaborations (BC). While the declarative and technology-independent nature of ebBP is beneficial for agreement, the implementation of B2Bi processes in the face of distributed and heterogeneous systems calls for a well-defined execution model. As a major integration technology, Web Services and WS-BPEL (BPEL)[4] have been proposed as implementation technology for ebBP BTs, e.g., [5] or [6]. Up to now, the aspects of composing BTs within BCs as well as the impact of QoS realization on control flow have not sufficiently been considered.

This paper investigates the implications of composition and QoS realization for the execution of BTs. The main contribution of the paper is a state-machine based execution model for ebBP BTs. The model is flexible, composable and QoS-aware as it allows for BT parameterizations, e.g., QoS, enables composition and reflects the implications of Web service based QoS realization. Although implementability and validity have been checked for Web services and BPEL by means of a prototype[2], the model is abstract in allowing for different integration styles or technologies.

Section 2 gives a short introduction to ebBP and BPEL; section 3 provides an overview of the integration architecture defining the background for the execution model. Subsequently, the requirements for the execution model as well as the execution model itself are presented in sections 4 and 5. After discussing related work in section 6, section 7 concludes and points out directions for future work.

2. Basics

ebBP and BPEL are core to our approach. Whereas the choreography language ebBP describes the interaction of integration partners from a global perspective, the

1. www.rosettanet.org

2. Available at http://www.uni-bamberg.de/pi/ebBP-BT-prototype.

orchestration language BPEL defines the partners' local implementation view.

ebBP is based on the concept of *BusinessTransactions* (BT) exchanging business documents (cf. section 4.1). So-called *BusinessCollaborations* (BC) with at least two roles (integration partners) can be used to build complex integration models. Control flow constructs like *Decision*, *Join* or *Fork* choreograph *BusinessTransactionActivities* (BTA) and *BusinessCollaborationActivities* (CA) that specify the actual execution of *BusinessTransactions* and *BusinessCollaborations*, respectively. BTAs and CAs add execution parameters such as *TimeToPerform* and map the roles of the performing *BusinessCollaboration* to the roles of the performed activity.

BPEL 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 or asynchronous interactions like *invoke*, *receive* or *onMessage* are defined and constructs like *sequence*, *if* and *while* are used to specify the control flow between these interactions. For details please refer to [4].

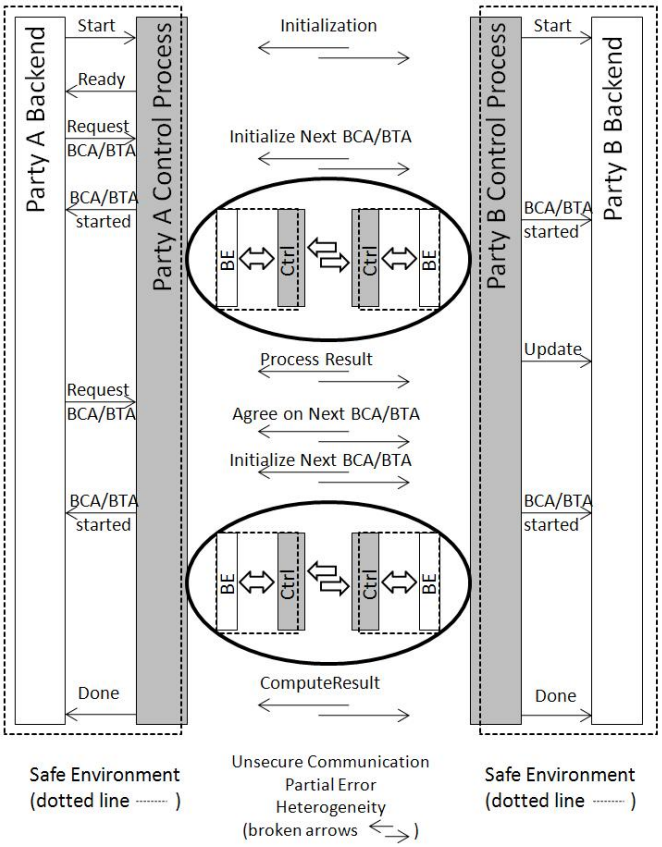

Figure 1. Modularized B2Bi integration scenario

3. Integration Architecture

Integration architecture plays an important role in the design of an ebBP BT execution model. In [7], 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 (implementations of ebXML's Business Service Interfaces) handle control flow and deal 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 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 BTA. The control processes then negotiate BTA parameters like an instance identifier or a time limit and pass on control to the lower level BTA 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, BTAs 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 principle setting for analyzing the requirements for an ebBP BT execution model.

4. Requirements Analysis

The core requirement for a BT execution model is to be conform with the ebBP specification of a BT that is presented in section 4.1. Additional requirements concerning result computation and realization of QoS, configuration options of the ebBP model as well as the containing BC's control flow are described in sections 4.2, 4.3 and 4.4.

4.1. The ebBP BusinessTransaction Model

An ebBP BT is a technology-agnostic specification of the exchange of a business document and an optional business document response between exactly two partners. So-called BusinessActivities (BA) specify each business document to be exchanged, together with additional business signals and business related QoS properties. Business signals, namely ReceiptAcknowledgement (RA) and AcceptanceAcknowledgement (AA), are designed to give the business document sender information about the state of a business document's processing by the receiver. A RA may indicate that a business document has been received by the partner's control process or, if the so-called *isIntelligibleCheckRequired* is flagged, that validation checks like structure or schema validation have been performed (cf. [3], section 3.4.9.3). An AA indicates that the business document has been accepted for business processing (cf. [3], section 3.4.9.3). For both, RA and AA, according ReceiptAcknowledgementExceptions (RAE) and AcceptanceAcknowledgementExceptions (AAE) are defined as well as a so-called GeneralException (GE, cf. [3], section 3.6.2.3) that covers all exceptional cases other than RAE and AAE.

QoS Attribute	Level of Specification
isAuthenticated	Doc
isConfidential	Doc
isTamperDetectable	Doc
isIntelligibleCheckRequired	BA
isNonRepudiationRequired	BA
isNonRepudiationReceiptRequired	BA
timeToAcknowledgeReceipt	BA
timeToAcknowledgeAcceptance	BA
isAuthorizationRequired	BA
retryCount	BA
isGuaranteedDeliveryRequired	BT
hasLegalIntent	BTA
isConcurrent	BTA
timeToPerform	BTA

Table 1. ebBP QoS attributes and specification levels

The exchange of business documents and signals is controlled by a set of business related QoS attributes as listed in table 1. Each attribute is associated with its level of specification, i.e., whether it applies to the business document (Doc), the BusinessActivity (BA), BusinessTransaction (BT) or BusinessTransactionActivity (BTA) that represents the execution of a BT within a collaboration. We consider most of the ebBP QoS attributes as self-explanatory, but *hasLegalIntent* and *isConcurrent* deserve additional explanation. For *hasLegalIntent*, ebBP states that *"The hasLegalIntent attribute could have widely differing interpretations and enforceability depending on type of business, process, and jurisdiction."* ([3], section 3.4.9.7). *isConcurrent* (cf. [3], 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. As a running example, listing 1 shows an ebBP example specification for exchanging a purchase order confirmation (POC) using a *DataExchange* business transaction type. The referenced business document is taken from RosettaNet[1].

Listing 1. RosettaNet based BT example

```
<DataExchange
 name="bt-PIP3A20"
 nameID="bt-PIP3A20"
 isGuaranteedDeliveryRequired="true">
 <RequestingRole name="POC sender"
  nameID="PIP3A20-role-sender"/>
 <RespondingRole name="POC receiver"
  nameID="PIP3A20-role-receiver"/>
 <RequestingBusinessActivity
  name="Send POC"
  nameID="PIP3A20-ba-req"
  isIntelligibleCheckRequired="true"
  isNonRepudiationRequired="true"
  isNonRepudiationReceiptRequired="true"
  retryCount="3"
  timeToAcknowledgeReceipt="PT3M"
  timeToAcknowledgeAcceptance="PT6M">
  <DocumentEnvelope
   name="PIP3A20 POC"
   businessDocumentRef="PIP3A20-POC"
   nameID="PIP3A20-POC-de"
   isAuthenticated="transient"
   isConfidential="transient"
   isTamperDetectable="transient"/>
  <ReceiptAcknowledgement
   name="ra" nameID="PIP3A20-ra"
   signalDefinitionRef="ra2"/>
  <ReceiptAcknowledgementException
   name="rae" nameID="PIP3A20-rae"
   signalDefinitionRef="rae2"/>
  <AcceptanceAcknowledgement
   name="aa" nameID="PIP3A20-aa"
   signalDefinitionRef="aa2"/>
  <AcceptanceAcknowledgementException
   name="aae" nameID="PIP3A20-aae"
   signalDefinitionRef="aae2"/>
 </RequestingBusinessActivity>
</DataExchange>
```

4.2. Mutual Agreement and QoS

Technically speaking, the very purpose of executing BTs is aligning state between integration partners. If a BTA is performed within a collaboration then the result of the BTA must be mutually agreed upon once the BTA execution has terminated. Otherwise, the state of integration partners may diverge. Clearly, the business implications of a BT execution depend on the business logic applied to the exchanged documents. As ebBP only defines technical aspects of BTs, the model used in this work defines the result of a BT execution as the *exchange state of business documents and business signals as well as fulfillment of ebBP QoS requirements*. Fulfillment of ebBP QoS requirements is necessary because

these may affect the exchange state of a business document. For example, if authentication is required then a business document must not be considered to have been successfully exchanged in case authentication information is missing. Furthermore, QoS attributes have to be implemented in a mutual way, i.e., both requester and responder must be authenticated, sure about integrity et cetera. Finally, QoS has to be equally applied to business documents and business signals because business signals influence the result of a BT execution. For example, using an unauthenticated RA to acknowledge legibility of an authenticated business document is not sensible.

4.3. Configurability

Different integration scenarios have different requirements with respect to state alignment and QoS. Transmitting RA and AA signals as well as defining strict security requirements may be required for the exchange of a purchase order while a RA may be sufficient for the exchange of a catalog. The different ebBP business transaction types (cf. [3] section 3.4.9.1) like *Notification*, *DataExchange* or *CommercialExchange* having different QoS needs evidence this fact. Business transaction types define restrictions upon the choice of BT parameters as described in section 4.1. For example, the *Notification* pattern requires using a requesting BA as well as a RA signal while the use of an AA as well as some QoS attributes are optional. The ebBP *DataExchange* pattern is the most flexible pattern in allowing either one or two BAs and defining no constraints upon the use of RAs, AAs or QoS parameters. In so far, the remaining ebBP business transaction patterns can be interpreted as instantiations of the *DataExchange* parameterization options. Therefore, supporting the *DataExchange* pattern is selected as a requirement for the ebBP BT execution model.

4.4. Composition Context

The invention of the *isConcurrent* parameter for ebBP BTs shows that its context, i.e., the way a BT is used within BCs, influences the BTs' implementation. Considering the integration setting described in section 3, the following requirements can be derived:

Concurrent execution At one point in time, more than one instance of a BT could be active.

Multiple execution Concurrent execution is one form of executing a BT several times. Also, a BT may be repeatedly executed in a loop, in sequence or in different control flow branches. This implies that routing of the collaboration needs more information than simply protocol success or failure. In order to enable flexible routing, a BT implementation should not only propagate a generic protocol outcome to the superordinate collaboration but also hand over the business documents exchanged.

Reconfiguration Multiple execution of BTs implies that the need for alignment messages and QoS attributes may vary as well. The ebBP model provides *isConcurrent*, *hasLegalIntent* and *timeToPerform* as configuration options for BTAs.

5. Execution Model

This section presents an execution model for ebBP BTs by defining state machines for BT control processes. The model is composable, flexible, QoS-aware and abstract.

Composability is facilitated in two ways. First, a consistent BT execution result with respect to the state of business document/signal exchanges and the fulfillment of QoS is ensured. Second, requirements emerging from the composition context as described in section 4.4 are respected. Furthermore, the execution model is *flexible* by allowing for almost arbitrary instance values of the *DataExchange* parameters. Also, the model is *QoS-aware* as the influence of QoS realization (sec. 5.1) on control flow (sec. 5.2) explicitly has been respected, most notably whether or not mutual agreement upon message delivery can be expected to be available at the messaging level (see below). In so far, section 5.1 only analyzes QoS realization up to the point that is decisive for building the control flow model. Finally, the execution model is *abstract* in not prescribing a concrete implementation of control processes. This is due to the fact that the implementation of control processes highly depends on the IT landscape of integration partners. Although a Web services and BPEL-based prototype has been implemented[2], partners may not want to implement control processes on top of BPEL or not even use Web services. Other communication technologies that meet the assumptions about QoS realization may be chosen as well.

5.1. Realization of Quality-of-Service

Realization of ebBP QoS requirements necessitates the combination of distributed algorithms targeting at security or reliability goals. Simply implementing a security layer on top of a reliability layer is not possible for two reasons:

1) Mutual realization of security-related QoS properties may create a new reliability problem. In practice, security properties like authentication or integrity are frequently implemented by means of attaching asymmetric digital signatures to the message payload. The receiver then can verify the authenticity of the sender, but the sender cannot be sure about authenticity of the receiver. Sending a signed acknowledgment with a hash value of the original message does not help because then the sender of the acknowledgment cannot be sure who has received the acknowledgment.

2) A malicious attacker must be assumed. A malicious attacker basically may try to manipulate any message exchanged between integration partners. This not only

holds true for the message payloads but for lower-level transport messages as well. This means that an attacker may try to manipulate communication by means of tampering with unsecured reliability messages. While an attacker may not be able to break arbitrary security goals, the reliability property is endangered.

Put short, mutual realization of security and reliability properties calls for algorithms that implement a secured reliable messaging layer.
The complexity of such algorithms raises the question which implementation components should perform QoS algorithms. In general, these algorithms may be implemented at the protocol level, i.e., by the control processes, or at the messaging level, i.e., by the messaging technology used for communication. Note that control processes are application level protocols. In practice, the complexity of distributed algorithms barely is acceptable at the application level.
Fortunately, realization of QoS at the messaging level is available for Web Services, especially the combination of reliable messaging and security. [8] and [9] both report the successful verification of the so-called "Secure WS-ReliableMessaging Scenario". In particular, [8] report successful verification of *"mutual authentication between client and service on all security-relevant messages."* If Web services are used, reliability (ebBP attribute *isGuaranteedDeliveryRequired*), confidentiality (*isConfidential*), integrity (*isTamperDetectable*) and authentication (*isAuthenticated*) therefore can be assumed to be implementable at the messaging level. The same holds true for authorization (*isAuthorizationRequired*) that can easily be implemented once that authentication is available.
Concurrency (*isConcurrent*) cannot be implemented at the messaging level as it concerns the separation of BT execution instances as well as synchronization of access to backend systems. If BPEL is used for control process implementation then BPEL correlations ([4], section 9) can be used for enabling concurrent control process instances. If not, WS-Addressing ([10]) could be used. Finally, if neither BPEL correlations nor WS-Addressing are appropriate a BPEL correlations like mechanism can be implemented at the application level. As regards the synchronization of access to backends we argue that functionality that is exposed to integration processes must be able to deal with concurrency. If necessary, the backends can be informed about the *isConcurrent* value by means of a flag.
hasLegalIntent cannot directly be implemented because its semantics are not clearly defined (cf. section 4.1). As far as BT control processes are concerned, integration partners may choose to map the *hasLegalIntent* value to different instantiations of other ebBP QoS attributes. Also, the *hasLegalIntent* value should be passed on to the backend applications.
isNonRepudiationRequired and *isNonRepudiationReceiptRequired* are special in implying a very hard error model. Non-repudiation usually is defined as the property that the sender of a particular message cannot deny having sent the message. The attempt to deny having sent/received a message implies that an integration partner cannot be assumed to behave as defined in a protocol specification. If so, the possibility of implementing two-way non-repudiation is questionable, i.e., the sender cannot deny having sent a message while the receiver cannot deny having received it. Consistently, no WS-* standard could be found that implements non-repudiation in a mutual way. Therefore, we propose to implement non-repudiation in an asymmetric way by simply attaching a signature to business documents and business signals and archiving these messages upon arrival. Once the non-repudiated message has been archived the receiver can claim to have successfully received the message. If a BT execution succeeds (which should be the standard case) then having implemented non-repudiation in an asymmetric way does not do any harm. If it fails, the receiver of the non-repudiated message may assert a claim based on the message. The remaining QoS features listed in table 1 can easily be implemented at the control process level and are discussed in the next section.

5.2. Control Flow

This section presents the control flow of control processes using a state machine based model. A core design decision is the choice of communication style between automata, i.e., synchronous or asynchronous. [11] find that assuming synchronous communication allows for easier automata design than asynchronous communication. Consistently, synchronous communication has frequently been assumed for model design and analysis (e.g., [12] [13] [14]). The last section spelled out that the realization of QoS properties also is possible for this communication style. Therefore, the work at hand also adopts a synchronous communication model. Note that synchronism as used here only concerns one single message exchange, i.e., the sender of a message only blocks until the message is delivered. Processing of the message is then performed asynchronously. Technically speaking, this corresponds to messaging with a buffer length of 0.
From a software engineering point of view, assuming synchronous communication between control processes imposes more strict requirements in terms of availability and throughput than asynchronous communication. This limitation can be justified because the control processes' only task is controlling the interaction between integration partners. At the same time, control processes can be used to implement decoupling between the integration partners' backend systems. For example, incoming business documents could be stored in a message queue by the control process and then be picked up by the backend when appropriate.

5.2.1. State Machine Model. This section describes the control processes for the running example of listing 1. Section 5.2.3 illustrates how to derive control processes for different instances of an ebBP BT model. For formalization, we are defining the notion of a *control process state machine* (CPS) as follows:

Definition 1 (control process state machine). A *control process state machine* (CPS) is a 7-tuple (S,s_0,F,E,I,G,δ) consisting of the following elements:

- S a finite set of states and $s_0 \in$ S the initial state.
- F $\subseteq$ S a non-empty set of final states.
- E = M$\cup$L a set of events where M is a set of message exchange events and L is a set of local events. Every m $\in$ M consists of a communication partner p, a direction d $\in \{!, ?\}$ and the actual message type t. We write p!t for an outgoing message t to partner p and p?t for an incoming message t from partner p.
- I a set of counters with domain $\mathbb{N}_0$.
- G a set of guards with G $\subseteq \mathbb{N}_0 \times \{<, \leq, =, >, \geq\} \times$ I
- δ is a partial transition function $\delta : \mathrm{S} \times \mathrm{E} \times 2^G \rightharpoonup \mathrm{S} \times 2^{\mathrm{I}}$

The following communication partners are used for the definition of CPSs that can be combined pairwise, i.e., the communication between two CPSs is not visible to a third CPS. REQ and RES denote the requestor and responder control process implementing an ebBP BT. BE_1 and MA_1 denote the backend and the master process of the requestor party where BE_1 implements business logic like validation of business documents while MA_1 implements the superordinate ebBP BC. Accordingly, there are BE_2 and MA_2 roles for the responder party. Additionally, RAC denotes a component implementing simple validation logic (e.g., schema validation) which is needed by the responder control process. The state diagrams depicted in figures 2 and 3 denote the transition relation of the REQ and RES CPSs. Note that the integration of control processes with backend systems is private logic of the integration partners and therefore is not a formal element of our ebBP BT execution model. Nonetheless, the BE_k, MA_k and RAC parties are included for showing one valid way of integration. Clearly, any implementation that accepts the same set of communication sequences between control processes is acceptable.

The message types (set M of a CPS) selected for interaction between control processes are aligned with the ebBP specification. *bizDoc* is an abstract type denoting the business document to be exchanged. *ra* and *aa* denote ebBP's ReceiptAcknowledgement and AcceptanceAcknowledgement, respectively, while *rae* and *aae* denote the according exceptions. *ge* denotes GeneralException that is used by ebBP for conveying exception information that is not covered by *rae* and *aae*. *start* (containing initialization information), *solBizDoc* (for soliciting the business document to be exchanged), *cancel* (for giving the backend the opportunity to cancel the BT run) and *persist* (for specifying that the

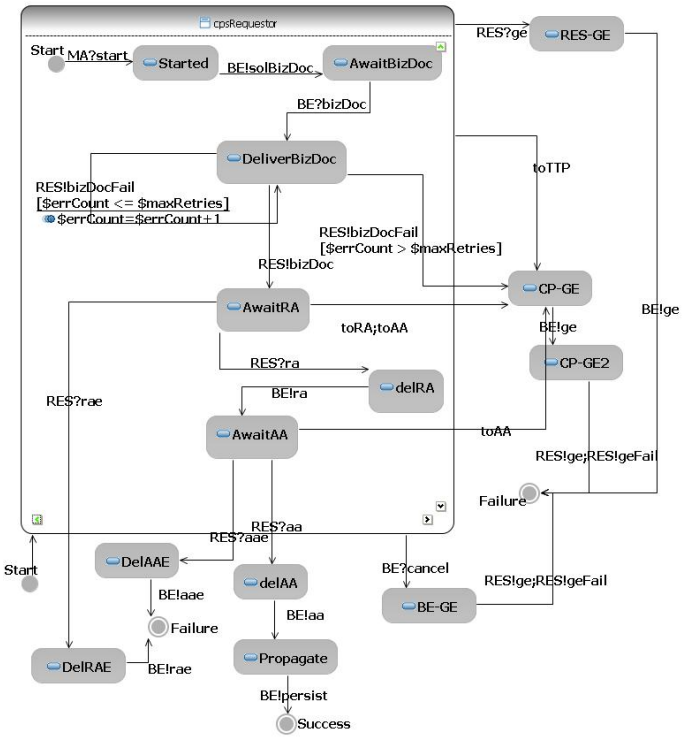

Figure 2. Requester Control Process Machine

message exchange succeeded) are only needed for modeling interaction with the remaining parties.

The following local events (set L of a CPS) have been defined. For any message type m $\in$ M, there is a <m>Fail event that models the case that the sending CPS could not deliver the message of type m to the receiving CPS. The event is modeled as local because the receiving CPS might not even realize that the messaging exchange failed. There are no <m>Fail events defined for the interaction between control processes and BE_k, MA_k, RAC processes because this interaction is assumed to be safe (cf. section 3). *toTTP*, *toRA*, *toAA* model timeout events for the *timeToAcknowledgeReceipt*, *timeToAcknowledgeAcceptance* and *timeToPerform* parameters of an ebBP BT definition. Basically, these timeout events are controlled by the REQ CPS in order to avoid concurrent timeouts. Only in the *AwaitBizDoc* state of the RES CPS, a *toTTP* event is defined in order to avoid that the RES CPS waits forever. I is defined as {errCount, maxRetries} for implementing ebBP's *retryCount* parameter. *isIntelligibleCheckRequired* is not explicitly reflected in the control flow. ebBP provides Receipt/AcceptanceAcknowledgements and the according exception types for conveying legibility information. The details of the validation checks required are to be defined by the integration partners. The *Success* and *Failure* states in figures 2 and 3 represent the state machines' final states. These states represent purely technical results, i.e., whether all necessary messages successfully have been exchanged or not. As a business transaction implementation may be

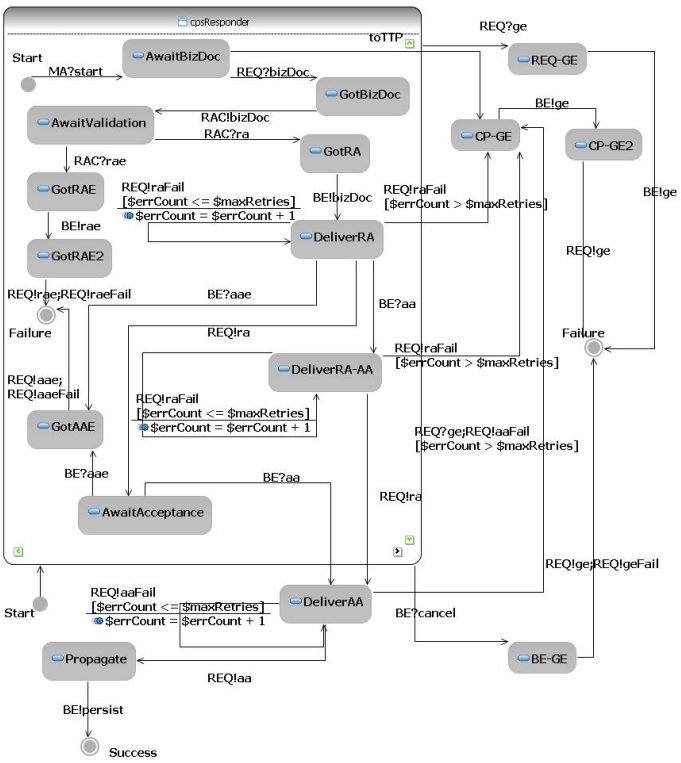

Figure 3. Responder Control Process Machine

reused in several contexts (cf. section 4.4), the computation of a business level result, e.g., whether or not a document exchange represents the obligation to pay an invoice, is to be done at the level of the calling collaboration implementation. This implies that the protocol result together with the exchanged business document are propagated to the calling implementation upon reaching a final state (not explicitly modeled in the state machines).

For presentation purposes, two visual simplifications have been introduced that are not reflected in the CPS definition. Composite states have been introduced for limiting the number of transitions. The actual automaton can be derived by adding a copy of each transition emerging from a composite state to every (non-pseudo-) state contained in the composite state. Also, when a transition carries two events separated by a semi-colon, then there actually are two transitions defined (and guards and increments only apply to the last event). Finally, note that the names of states do not contribute to the semantics of the state machines and therefore are not explicitly explained.

5.2.2. Semantics. Common state machine semantics do not assume a synchronous communication model (like in [15]) or do not consider local events (like in [13]). Therefore, we define the semantics of CPS operationally as follows (remember that the model definition is non-hierarchical):

Consider a pair of CPS machines (CPS_1,CPS_2). Let M_p be the set of all message types that can be exchanged between CPS_1 and CPS_2. Further, let a configuration c be defined as $c \in \mathcal{C} = S_1 \times E_1^* \times V_1 \times S_2 \times E_2^* \times V_2$ where V is the set of all possible values of the counters of a CPS: $V = 2^{I \to \mathbb{N}^0}$. Finally, let $\psi : G \times V \to \{tt, ff\}$ be the boolean function that evaluates guards under given counter values to true (tt) and false (ff) and ϕ the function that extracts the communication partner from a messaging event. The semantics of a pair of CPS machines CPS_1 and CPS_2 is then defined by the relation $\vdash$ on $\mathcal{C} \times \mathcal{C}$ that captures communication between automata (*comm*) or local behavior of one automaton (*local*). Rules 1 and 2 describe a message exchange from CPS_1 to CPS_2 and a local event of CPS_1. For brevity, symmetric rules for CPS_2 are left out:

1: $(s_1, CPS_2!m_1v_1, v_1, s_2, CPS_1?m_2v_2, v_2) \vdash^{comm}$
$(s_1', v_1, v_1', s_2', v_2, v_2') \quad iff$
$m_1 = m_2 \in M_p \wedge$
$\delta(s_1, CPS_2!m_1, \alpha_1) = (s_1', \beta_1) \wedge$
$\alpha_1 = \{\} \wedge \forall i \in \beta_1 : v_1'(i) = v_1(i) + 1 \wedge$
$\delta(s_2, CPS_1?m_2, \alpha_2) = (s_2', \beta_2) \wedge$
$\alpha_2 = \{\} \wedge \forall i \in \beta_2 : v_2'(i) = v_2(i) + 1$

2: $(s_1, e_1v_1, v_1, s_2, e_2v_2, v_2) \vdash^{local}$
$(s_1', v_1, v_1', s_2, e_2v_2, v_2) \quad iff$
$(e_1 \in L_1 \vee \phi(e_1) \neq CPS_2) \wedge$
$\delta(s_1, e_1, \alpha_1) = (s_1', \beta_1) \wedge$
$\forall g \in \alpha_1 : \psi(g, v_1) = tt \wedge$
$\forall i \in \beta_1 : v_1'(i) = v_1(i) + 1$

5.2.3. Configurability. Section 4 mandates that the ebBP BT execution model should be freely configurable. The implementation of the execution parameters as listed in table 1 has been discussed above. Yet, there could be different selections of control messages, i.e., whether Receipt/AcceptanceAcknowledgements are to be used or if a second business document is to be exchanged. The correct CPSs when leaving out a Receipt/AcceptanceAcknowledgement can be derived by first removing all transitions triggered by an *ra/rae/aa/aae* and then removing all states that do not have outgoing transitions triggered by message exchanges anymore and then adjusting dangling transitions accordingly. For example, if RA and AA would have to be removed from the REQ CPS, the *RES!bizDoc* transition would have to be connected to the *Propagate* state. The correct REQ CPS for an ebBP two-action BT can be derived by appending the RES state machine to the REQ state machine. When appending, the *Start* state of the RES CPS replaces the *Propagate* state of the REQ state machine. The two-action RES state machine then can be derived accordingly. Note, that the responsibility for monitoring the overall *timeToPerform* then must be switched from the REQ CPS to the RES CPS after finishing the first action.

Configurability also raises the question of dependencies between configuration parameters. For example, defining a *timeToAcknowledgeReceipt* requires that a ReceiptAcknowledgement is exchanged. Due to space limitations, a thorough discussion of dependencies cannot be provided here.

6. Related Work

Abstract models for interacting parties are frequently researched, e.g., [16], [13] or [17]. In our work we specifically target at ebBP BusinessTransactions that cover domain-specific characteristics of B2Bi document exchanges. In this area, the execution of RosettaNet PIPs [18], [19] or UMM BusinessTransactions [5], [20], [21] using Web Services or WS-BPEL has been researched. RosettaNet PIPs as well as UMM BusinessTransactions share a lot of concepts with ebBP BusinessTransactions. The work presented there does not investigate the implications of composing BusinessTransactions within collaborations in detail. Particularly, implementation of QoS attributes in a mutual way has not been considered. A precise abstract model with clearly defined semantics based on synchronous communication also has not been presented. The same holds true for [6] that describes the execution of ebBP BusinessTransactions using BPEL processes while respecting QoS. In particular, QoS attributes are implemented asymmetrically only.

7. Conclusion and Future Work

In this work we have presented an ebBP BusinessTransaction execution model that reflects the implications of composing BTs within BCs, that is abstract in allowing for different implementations of control processes, and that is flexible in allowing for different instantiations of an ebBP model. A BPEL-based prototype[2] implementation has been used for validating the model.
Future work comprises implementing the execution model in the context of an model-driven approach that derives distributed orchestrations from ebBP choreographies as described in [7] as well as validating such a model-driven approach in an industry case study. Further, the applicability of the execution model to other important B2B integration technologies like AS2 and ebMS is to be investigated.

References

[1] Backer et al., "A scenario-based verification technique to assess the compatibility of collaborative business processes," *Data Knowl. Eng.*, vol. 68, no. 6, pp. 531–551, 2009.

[2] A. Schönberger and G. Wirtz, "Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions," in *SERP 2006*.

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

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

[5] 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.

[6] A. Schönberger et al., "QoS-enabled business-to-business integration using ebBP to WS-BPEL translations," in *Proc. of the IEEE SCC 2009 Int. Conf. on Services Computing*.

[7] A. Schönberger and G. Wirtz, "Using variable communication technologies for realizing business collaborations," in *Proc. of IEEE 2009 World Congress on Services (SERVICES PART II)*.

[8] M. Backes, S. Moedersheim, B. Pfitzmann, and L. Vigano, "Symbolic and cryptographic analysis of the secure ws-reliablemessaging scenario," in *FOSSACS 2006*.

[9] K. Bhargavan, R. Corin, C. Fournet, and A. D. Gordon, "Secure sessions for web services," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, p. 8, 2007.

[10] M. Gudgin, M. Hadley, and T. Rogers, *Web Services Addressing 1.0 - Core*, W3C, May 2006.

[11] T. Bultan, J. Su, and X. Fu, "Analyzing conversations of web services," *IEEE Int. Comp.*, vol. 10, no. 1, pp. 18–25, 2006.

[12] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 292–333, 1997.

[13] B. Benatallah, F. Casati, and F. Toumani, "Representing, analysing and managing web service protocols," *Data Knowl. Eng.*, vol. 58, no. 3, pp. 327–357, 2006.

[14] S. McIlvenna, M. Dumas, and M. T. Wynn, "Synthesis of orchestrators from service choreographies." in *Proc. of 2009 Asia-Pacific Conf. on Conceptual Modelling (APCCM)*.

[15] R. Eshuis, "Reconciling statechart semantics," *Sci. Comput. Program.*, vol. 74, no. 3, pp. 65–99, 2009.

[16] W. M. P. van der Aalst and M. Weske, "The P2P approach to interorganizational workflows," in *Proc. of the 2001 Int. Conf. on Advanced Information Systems Engineering (CAISE)*.

[17] T. Bultan and X. Fu, "Specification of realizable service conversations using collaboration diagrams," *Service Oriented Computing and Applications*, vol. 2, no. 1, pp. 27–39, 2008.

[18] R. Khalaf, "From RosettaNet PIPs to BPEL processes: A three level approach for business protocols," *Data Knowlegde Engineering*, vol. 61, no. 1, pp. 23–38, 2007.

[19] A. Schönberger and G. Wirtz, "Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study," in *2006 Int. Conf. on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing (EEE)*.

[20] C. Huemer and M. Zapletal, "A state machine executing UMM business transactions," in *Proc. of the 2007 IEEE Int. Conf. on Digital Ecosystems and Technologies (DEST)*.

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