Introducing Partner Shared States into ebBP to WS-BPEL Translations

Christoph Pflügler
Inter-organizational Systems Group
University of Augsburg
Augsburg, Germany
christoph.pfluegler@wiwi.uni-augsburg.de

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

Guido Wirtz
Distributed and Mobile Systems Group
University of Bamberg
Bamberg, Germany
guido.wirtz@uni-bamberg.de

ABSTRACT

Business-to-Business integration (B2Bi) as a core concept of Supply Chain Management (SCM) is a key success factor for enterprises today. Frequently, choreography models are used for agreeing about the overall message exchanges among integration partners, while orchestration models are used to specify the local message flow of each individual participant. Choreography standards like ebXML BPSS (ebBP) and orchestration standards like WS-BPEL have been developed which promise to provide standards based support for interoperability among integration partners. Further, the translation of ebBP models to WS-BPEL models has been proposed in order to ensure, among others, conformance of orchestrations to choreographies.

This paper focuses on the agreement function of choreography models and introduces the concept of *partner-shared-states* to ebBP models in order to better capture the effect of business document exchanges. The translation of a restricted set of partner-shared-state based ebBP models to WS-BPEL has been implemented in order to prove the feasibility of the approach. The resulting WS-BPEL processes are used to guarantee an order of message exchanges that is choreography-compliant, while a backend services interface encapsulating business logic is used for providing the control process with business documents, business decisions and events. The overall approach is evaluated using a RosettaNet PIP based use case.

Keywords

B2Bi, ebXML BPSS, WS-BPEL, state based modeling, translation

iiWAS2009 December 14–16, 2009, Kuala Lumpur, Malaysia

1. INTRODUCTION

In today's competitive world, the success of an enterprise heavily depends upon effective integration along the enterprise's supply chain (cf. [8]). B2Bi as a core task of SCM (cf. [4]) therefore deserves special attention by industry and academia. B2Bi particularly addresses the integration of processes crossing enterprise boundaries where central IT infrastructure for integration partners frequently is not available. Further, personnel from different enterprises with differing background and terminology is participating. Challenging tasks concerning various forms of *consistency* result from this situation. According to [16], tasks such as agreement among integration partners upon the type and order of business document exchanges, compatibility between each integration partner's local process definition with respect to these exchanges, interoperability concerning the communication technologies applied as well as synchronizing state between IT systems are of paramount importance. One way to tackle these problems is using a so-called choreography language for capturing the global message exchanges and then translating the choreography model into participant specific orchestrations that define the types and sequence of the messages a single integration partner is supposed to receive and send. While a choreography model can be used as means for agreement, a sound derivation of orchestration models from a choreography model is one means to ensure compatibility between the partner's local processes. Interoperability of IT systems heavily depends on the communication technology applied whereas state synchronization can either be realized by using suitable communication primitives or by implementing distributed commit protocols on an application level.

In practice, one choreography language of choice is ebBP [12] that particularly focuses on business collaborations. ebBP is an official OASIS[1] standard and thus not only serves for specifying models to agree on but also supports communication among integration partners by defining a common vocabulary and common rules for defining choreographies. Regarding communication technology, the goals of decoupling systems, realizing interoperability and using com-

[1] `http://www.oasis-open.org`

monly available Internet protocols can be achieved by applying Web Services. In the area of Web Services based B2Bi, WS-BPEL [13] currently is the most widespread standard for specifying the process orchestration of each partner. ebBP defines so-called *BusinessTransactions* for exchanging business documents and proposes a protocol based on transmission notifications for performing *BusinessTransactions*. Hence, implementing this protocol in WS-BPEL (BPEL for short) is one way to achieve state synchronization. In this scenario, automatically deriving BPEL orchestrations from ebBP choreographies ensures compatibility between each partner's local process and reduces development time as well. Accordingly, translating ebBP into BPEL already has been proposed in literature [7].

This paper adopts the B2Bi approach of translating ebBP choreographies into BPEL orchestrations, but particularly focuses on the applicability of so-called *shared states* as introduced in [15]. *Shared states* support the agreement function of choreography models by capturing the effect of business document exchanges, allow for the definition of state specific timeouts, provide natural synchronization points in the collaboration's control flow and define the basis for intelligible communication of the collaboration's progress. Section 2 first describes how *shared states* can be represented in ebBP in a standard compliant way. A restricted set of ebBP models is then defined for which the translation to BPEL is to be investigated. Section 3 introduces an example of such a restricted model that is also used to evaluate the approach later on. In section 4, the proposed integration architecture for executing BPEL processes is described whereas the translation of ebBP to BPEL is discussed in 5. In doing so, this paper sets out to explore the practical feasibility of the approach rather than proving its correctness. The BPEL processes resulting from translation are validated by checking their executability on the Apache ODE[2] BPEL engine. Section 6 presents results as well as practical experience in implementing the translation. Finally, section 7 discusses related work and section 8 the contributions of this paper as well as future work.

2. ebBP MODEL

This section gives a short introduction to ebBP and then motivates the introduction of the *shared state* concept as well as the related concept of *micro-choreographies* to ebBP. Sections 2.1 and 2.2 then describe how these concepts can be represented in ebBP in a standard compliant way so that standard tools can be used for modeling. Section 2.3 finally describes the composition of these concepts in ebBP collaborations and informally defines the range of ebBP models that can be translated using our approach (see section 5).

ebBP is a choreography language based on the concept of *BusinessTransactions*[3] that are used for exchanging one or two business documents between the so-called *RequestingRole* and *RespondingRole*. So-called *BusinessCollaborations* with at least two roles (integration partners) can be used to build complex integration models. Therefore, usual control flow constructs like *Decision*, *Fork* or *Join* can be used to choreograph *BusinessTransactionActivities* (BTA) / *BusinessCollaborationActivities* (CA) that specify the actual execution of *BusinessTransactions* / *BusinessCollaborations* by adding execution parameters such as *TimeToPerform* and by mapping the roles of the performed *BusinessCollaboration* to the roles of the performed activity. Finally, QoS properties may be specified for the elements described and *BusinessDocument* definitions can be imported from business document libraries like RosettaNet[4] or UBL[5].

As ebBP models are the basis for agreement upon the choreography among the integration partners of a B2Bi, support of communication among personnel of the integration partners is of paramount importance. Therefore, the concept of so-called *shared states* commonly left/reached by all integration partners using so-called *micro-choreographies* as introduced in [15] is applied to ebBP modeling. This concept is based on the perception that the purpose of a business collaboration is essentially the consistent change of state of each integration partner's systems. Thus explicitly modeling these *shared* states helps in specifying the actually intended effect of the overall collaboration as well as the intended effect of exchanging business documents. The concept of micro-choreographies helps in abstracting from several business document exchanges that commonly lead to the change of a shared state and eases the handling of communication and processing errors. For example, assume a quote and an order are to be exchanged within a B2Bi and that at the beginning of the collaboration neither a quote nor an order have been exchanged. Then it would be the purpose of a micro-choreography to switch this shared state *Start* to the next shared state *Quote* by exchanging a quote request and quote document. The *Quote* state would then indicate that a valid quote is present but not (yet) an order. An order could be existent in the next state *Order* which, again, would be reached by means of a micro-choreography exchanging one or more business documents. Note that, using shared states, the integration partners could easily specify whether the exchanged quote should still be valid after having reached the shared state *Order*. Further, shared states are required to specify certain types of timeouts, e.g., how long a quote should remain valid, and which states are to be switched to when timeouts or errors occur during the course of micro-choreographies (cf. [15]). A possible scenario taking advantage of that feature is a supplier reserving resources, such as raw materials, when sending a quote to a customer and subsequently entering a shared state Quote (cf. section 3). The expiration time of the quote exchanged can easily be specified as timeout of the shared state Quote. If the customer does not respond to the quote before the timeout occurs, the overall collaboration could, e.g., switch to a shared state Failure and the supplier would be allowed to release the resources reserved. Also, modeling shared states explicitly introduces natural synchronization points to both, choreography and orchestration models of business collaborations. On the choreography level, this results in more compact models of collaborations with complex control flow. This can be seen by translating the use case of section 3 into a model that does not make use of shared states (omitted here due to space limitations). On the orchestration level, this results in handy reference points for attaching control flow logic. Finally, shared states define the basis for intelligible communication of the collaboration's progress which may be used for notifying process stakeholders.

[2]http://ode.apache.org/

[3]In this section, ebBP keywords are emphasized.

[4]http://www.rosettanet.org

[5]http://www.oasis-open.org/committees/ubl/

2.1 Modeling micro-choreographies

Micro-choreographies are a means for aggregating isolated document exchanges into units of correlated document exchanges that together lead to a shared state change. ebBP *BusinessTransactions* are a natural fit for representing micro-choreographies of up to two business document exchanges: *BusinessTransactions* add accompanying business signals to business document exchanges, and a standard way for computing the status of a BTA is defined in ebBP (cf. [12] sec. 3.6.3).

For the purpose of the work at hand, i.e., showing the feasibility of translating shared state based ebBP models into BPEL orchestrations, the restriction of micro-choreographies composed of only two document exchanges is accepted and thus *BusinessTransactions* are chosen for representing micro-choreographies. Note that "multi-document" micro-choreographies could be modeled in ebBP using *BusinessCollaborations* which then could be composed using CAs.

2.2 Modeling shared states and timeouts

There is no ebBP construct that directly matches the concept of a shared state so these are emulated. Generally speaking, a state can be modeled with an ebBP *Join* construct followed by a *Fork* construct. However, ebBP prohibits directly linking *Joins* and *Forks* as the corresponding *fromBusinessStateRef* and *toBusinessStateRef* attributes may only reference BTAs or CAs (cf. [12] sec. 3.8.2). To overcome this constraint in a standard compliant manner, the concept of an *EmptyBTA* based on the extensible ebBP transaction type *DataExchange* is introduced that serves as a target for linking to a shared state and for connecting the *Join* and *Fork* of a shared state.

Listing 1 shows the ebBP representation of the shared state *Quote* (cf. above and figure 1). The *EmptyBTA* before the shared state's *Join* is used as target of ebBP *Decisions* that are not allowed to directly link to *Joins* (cf. [12] sec. 3.8.2). The shared state's *Join* links to another *EmptyBTA* that is connected to the shared state's *Fork*. This *Fork* then specifies a *ToLink* for every micro-choreography that is permissible to be performed from this shared state. Shared state timeouts, i.e., the point in time when shared states should be left without performing a BTA, can also be specified on this *Fork*. In case such a timeout occurs, it has to be switched to the *corresponding Join* as required by ebBP (cf. [12] sec. 3.4.11.1).

As the exact semantics of a *corresponding Join* is not defined in ebBP, the work at hand computes the *corresponding Join* as the first common *Join* that can transitively be reached by all *ToLinks* of all *Decisions* attached to all BTAs the considered *Fork* links to. This is one of the reasons why a shared state is not simply modeled as an *EmptyBTA* with a following *Fork* but uses a *Join* and another *EmptyBTA*.

Employing two *EmptyBTAs* allows for different semantics when linking to a shared state with respect to timeouts: In case of linking to the *EmptyBTA* before a shared state, its timeout is reset whereas linking to the *EmptyBTA* within the shared state does not have this effect. The latter case is particularly useful if protocol failures occur during performing a subsequent BTA which means that the shared state actually has not been left. Note that ebBP *Joins* and *Forks* are only used for modeling states and are not allowed elsewhere in the collaboration description.

Listing 1: Example ebBP Model of a Shared State

```

<!-- State Quote -->
<BusinessTransactionActivity
    businessTransactionRef="empty"
    nameID="empty_before_Quote">
 <TimeToPerform></TimeToPerform>
 <Performs currentRoleRef="Buyer"
     performsRoleRef="empty1"/>
 <Performs currentRoleRef="Seller"
     performsRoleRef="empty2"/>
</BusinessTransactionActivity>

<Join waitForAll="false" nameID="Quote">
 <FromLink fromBusinessStateRef="
     empty_before_Quote"/>
 <FromLink fromBusinessStateRef="
     empty_before_Quote"/>
 <ToLink toBusinessStateRef="
     empty_in_Quote"/>
</Join>

<BusinessTransactionActivity
    businessTransactionRef="empty"
    nameID="empty_in_Quote">
 <TimeToPerform></TimeToPerform>
 <Performs currentRoleRef="Buyer"
     performsRoleRef="empty1"/>
 <Performs currentRoleRef="Seller"
     performsRoleRef="empty2"/>
</BusinessTransactionActivity>

<Fork nameID="fork_Quote" type="XOR">
 <TimeToPerform duration="P3D"></TimeToPerform
     >
 <FromLink fromBusinessStateRef="
     empty_in_Quote"/>
 <ToLink toBusinessStateRef="
     BTA_3A10_NotifyOfQuoteAck"/>
 <ToLink toBusinessStateRef="
     BTA_3A10_NotifyOfQuoteAck"/>
</Fork>
```

2.3 Composing micro-choreographies and shared states

This section first describes the interplay of micro-choreographies and shared states and then characterizes the set of ebBP collaborations that is used for checking the practicability of the ebBP to BPEL translation approach.

Basically, a shared state is entered by reaching the *EmptyBTA* before the shared state. The *Fork* of the shared state then links to all BTAs (micro-choreographies) that are permissible for the respective shared state. Each of these BTAs (not *EmptyBTAs*) must be followed by an ebBP *Decision* that evaluates the outcome of the BTA. Predefined ebBP *ConditionGuardValues* and user-defined *DocumentEnvelopes* are used for determining the follow-on shared state of a *Decision*. In case an ebBP *AnyProtocolFailure* is detected, the *Decision* typically links back to the *EmptyBTA* within the shared state the BTA to be evaluated was started from. Otherwise, it is linked to the *EmptyBTA* before (the same or another) a shared state. In general, automated model translation requires accepting syntactic restrictions (cf. [6]). The work at hand targets at evaluating the incorporation of shared states into ebBP2BPEL translations of real-world size collaborations. Balancing implementation effort of the ebBP to BPEL translator and expected benefit for the evaluation of the approach, the set of ebBP models that is considered for translation is a subset of the class of multi-transmission interactions as defined in [2] with the special restriction that only two collaboration partners are allowed:

- A choreography is modeled as a single ebBP *Business-Collaboration*. Hierarchical compositions are not supported.
- Only binary collaborations are supported, i.e., the number of integration partners within the collaboration is limited to two.
- A collaboration starts with an ebBP *Start* that immediately links to the initial shared state of the collaboration.
- ebBP *Decisions* are only allowed after BTAs.
- Alternative paths are realized by ebBP *Decisions* and, by ebBP *Forks* used for representing shared states.
- Looping is realized by *Decisions* that link back to shared states that have been visited before.
- The only case in which a *Decision* branch does not link to a shared state is when process termination is detected. In this special case a *Decision* links to an EmptyBTA before an ebBP *Success* or *Failure* state.
- A choreography ends when a final state, i.e., an ebBP *Success* or *Failure* state is reached. Multiple *Success* and *Failure* states are allowed per collaboration. As multiple instances of BTAs are not allowed for and as state is synchronized after each BTA (there are only two participants), a choreography immediately ends when a final state is reached.
- At any point in time, there is at most one active BTA (no multiple instances). In order to ensure this, ebBP *Forks* have to set the *type* attribute to *XOR* and ebBP *Joins* must have the *waitForAll* attribute set to *false*.
- An EmptyBTA may only link to one single ebBP *Join* or *Fork* element.

Note that ebBP does not define its execution semantics formally. Therefore, this work realizes the message flow of *BusinessTransactions* as described in section 4.1 and then applies this message flow iteratively as defined by the links between shared states, BTAs (micro-choreographies) and *Decisions*. Additional messages then are only needed for informing backend implementations about the current collaboration state and for deciding which BTA to execute in case multiple BTAs are triggered concurrently. By translating ebBP collaborations as defined above to BPEL this work provides an operational semantics of ebBP.

3. USE CASE

In general, our approach targets at B2Bi scenarios that use a transactional form of business document exchanges leading to shared state changes. In so far, compositions based on document exchanges of arbitrary document libraries such as RosettaNet, UBL or NES can benefit from the work presented here. The use case for evaluating the work at hand is based on RosettaNet *Partner Interface Processes* (PIPs) which describe the application context, the content and the parameters for the electronic exchange of one or two business documents. A use case consisting of nine shared states and nine PIPs has been created covering the concepts described above. The RosettaNet document type definitions have been imported by means of ebBP *BusinessDocumens* and their flow has been remodeled using ebBP *BusinessTransactions*. The use case is taken from RosettaNet PIP segment 3A and models a process for negotiating a contract. The size of standard processes as defined by the Northern European Subset[6] (NES) is comparable to our use case, so the use case's size can be considered to be realistic.

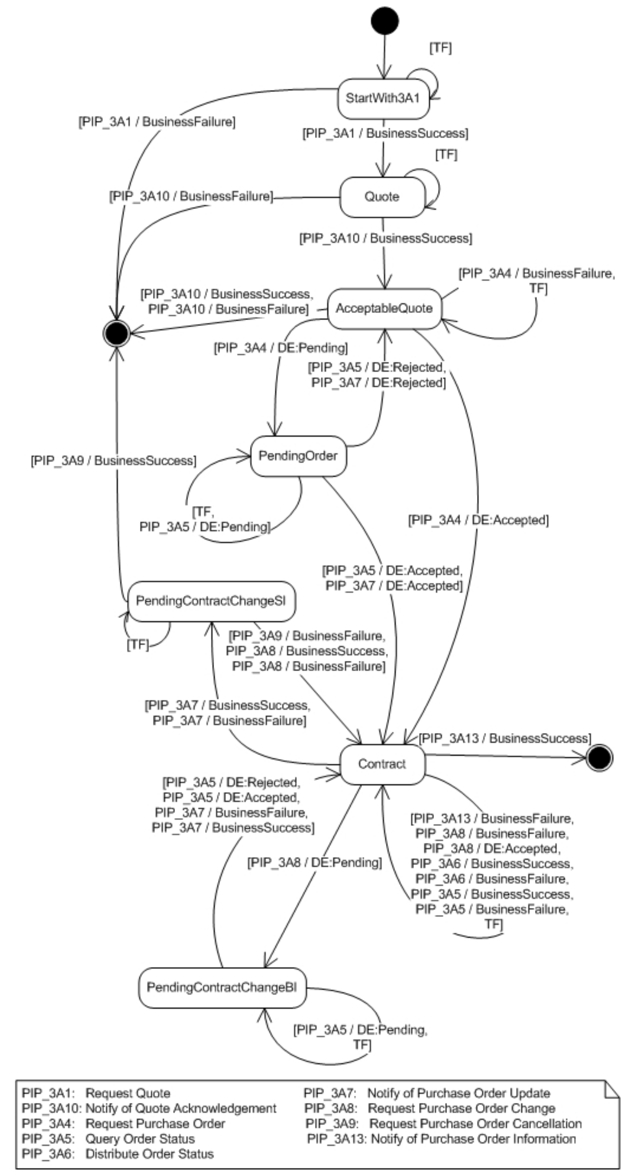

Figure 1: Use case for evaluating the work at hand

An excerpt of the use case is given as an informal state machine diagram in figure 1. The start of the collaboration is represented by the unique start element. Each shared state is represented as a state and the executions of PIPs as BTAs are represented as transitions. The event part of a transition is used to name the BTA (PIP) to be executed and the guard part of a transition is used to capture the outcome of BTAs. As decisions are not explicitly visualized, there may be multiple transitions for the same shared state that are triggered by the same event. The condition guards of the particular transitions, however, are mutually exclu-

[6] `http://www.nesubl.eu/`

sive. The permissible ebBP guard values for the use case are *AnyProtocolFailure* (denoted `TF`), *BusinessFailure* or *BusinessSuccess*. *AnyProtocolFailure* captures arbitrary technical problems during performing BTAs. If no such problems occur, *BusinessSuccess* indicates that integration partners did achieve their goals from a business point of view whereas *BusinessFailure* indicates they didn't. Finally, guard values based on DocumentEnvelopes (denoted with a leading `DE:`) that relate to the content of the latest business document exchanged using suitable XPath expressions are allowed as well. Two final states are used to represent an ebBP Failure state (on the left-hand side) and an ebBP Success state (on the right-hand side). Although the execution of `PIP_3A9` in state `PendingContractChangeSI(SellerInitiated)` may terminate with a *BusinessSuccess* guard value, it still represents a failure from the overall collaboration perspective. Note that state changes because of timeouts are not visualized.

4. INTEGRATION ARCHITECTURE

This section describes the proposed integration architecture for realizing B2Bi because it heavily influences the derivation of BPEL orchestration models from ebBP choreographies. The application of one BPEL process per integration partner, as opposed to applying a single central BPEL process, is proposed because central IT infrastructure is assumed not to be available by integration partners or simply not intended. According to [17] this solution (apparently) scales better than using one single BPEL process and therefore seems to support a broader range of B2Bi scenarios. Further, B2Bi projects usually have to consider the investments of integration partners in existing IT infrastructure and therefore have to address the problem of interfacing with existing systems. If a B2Bi project simply automates an existing process then there is a high probability that integration partners already have systems in place for evaluating business documents, taking business decisions and capturing real-world events such as "a new order has to be placed".
Therefore, the application of so-called *control processes* that separate the message flow of a collaboration from the actual business logic is proposed. It's the control processes' task to ensure that the message flow at runtime conforms to the choreography defined. The actual business logic is encapsulated in so-called backend services that wrap existing systems. This separation of concerns is also advantageous in terms of software lifecycle management because the integration partners' processes can be generated such that they do not have to be adapted after generation. This approach is also applicable for an integration partner that does not yet have systems implementing business logic. Note, that this work focuses on the message flow among the control processes and backend services while, clearly, there's much more to a B2Bi project, e.g., data mappings and adaptations of business functions.

4.1 Message Flow

The core task in describing the message flow between control processes and backend services is the mapping of BTAs. The flow of *BusinessCollaborations* can then be derived by repeating the message flow of the respective BTAs according to the ebBP choreography.
Figure 2 uses a UML sequence diagram to show an idealized flow of a BTA that exchanges two documents and employs both ebBP *ReceiptAcknowledgements* (RA) and *AcceptanceAcknowledgements* (AA) as accompanying business signals. BTAs that only exchange one business document or do not employ business signals can be mapped analogously. Figure 2 distinguishes between a *requesting role* for the in-

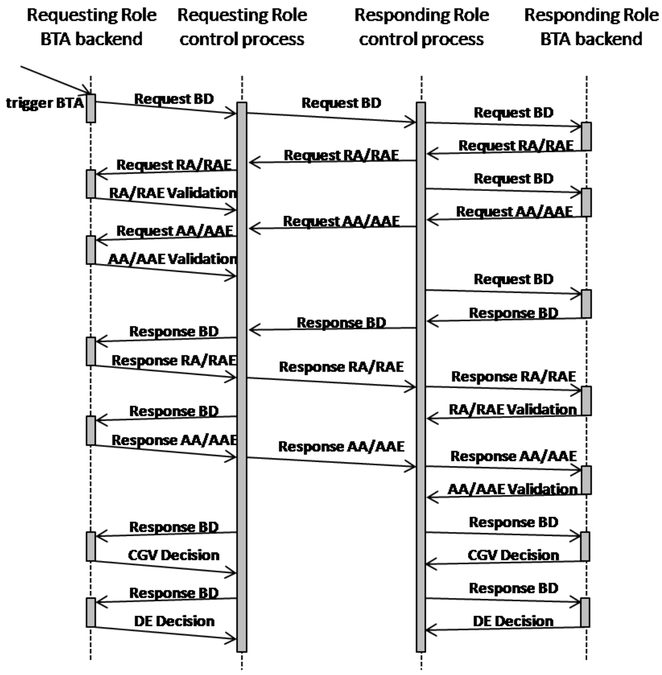

Figure 2: Idealized message flow of a BTA

tegration partner who sends the first business document of the BTA and a *responding role* for the sender of the reply business document. The message flow of the BTA starts out with the backend of the requesting role capturing the real world event that a new BTA has to be performed and thus sends the request *BusinessDocument* (BD) to the requesting role's control process. The latter then passes the Request BD on to the responding party's control process that subsequently sends the BD to the responding party's backend services for obtaining a RA and an AA or the corresponding exceptions (RAE and AAE). These business signals are then sent back to the requesting role's control process for indicating that the request BD is readable and has been accepted for business processing (cf. [12]). The same procedure is afterwards performed for the response BD using exchanged roles.
After having exchanged all business documents and business signals, both control processes call their backend services for evaluating the outcome of the BTA according to the messages exchanged. Clearly, each integration partner has to apply the same evaluation rules agreed upon in the ebBP choreography. Therefore, the work at hand employs ebBP *ConditionGuardValues* (CGV) and ebBP *DocumentEnvelopes* (DE), though for many business collaborations more sophisticated means will be necessary. A possible solution may be the definition of Schematron[7] files and thus it is assumed that such an agreement can be made.
Apart from deciding which shared state to switch to after a BTA, state changes have to be performed. We propose

[7] `http://www.iso.org/PubliclyAvailableStandards`

that such state changes are not performed until the end of a BTA. In order to perform state changes that are consistent among integration partners, distributed agreement has to be achieved. The realization of a BTA makes a step towards distributed agreement by applying business signals for excluding some error cases, but some business scenarios may require true distributed commitment. Though this is not yet implemented there are solutions to this problem available. One solution is to simply map the well-known Two-Phase-Commit protocol (2PC) to Web Services where the subject of agreement would be that all BTA messages have been exchanged (see [14] for details). Alternative solutions could be based on standards such as WS-ReliableMessaging v1.2[8] or Web Services Transaction v1.2[9].

4.2 BPEL and WSDL Artifacts

As pointed out above, this work proposes the generation of BPEL processes for implementing control processes and WSDL interfaces for encapsulating business logic. Especially the WSDL files for backend services may contain sensitive information, e.g., endpoint references, that should be hidden from the integration partner. Therefore, the structure of BPEL and WSDL files as depicted in figure 3 is proposed. Horizontal gray bars represent WSDL file types, the black squares in such a gray bar represent multiple copies of the same WSDL file. The vertical bars without filling show WSDL-files grouped together in sub packages, either for the purpose of providing a backend interface or a control process.

An ebBP business collaboration results in one BPEL process (RoleX.bpel) per participating party and several WSDL interfaces. common_msg_state.wsdl contains the definition of the collaboration's shared states and defines a WSDL `message` for communicating these. stateReceiverX.wsdl imports common_msg_state.wsdl and moreover defines the WS DL portType as well as the service definition and partnerLinkType of the Web Service (one per participating party) used for notifying the backend about the current process state. In figure 3, the two grey bars denoted stateReicev-erX.wsdl represent the same WSDL file except for the port addresses that are used for signaling process states which are partner specific.
Further, for each BPEL process, RoleX.wsdl and RoleX_back end.wsdl are defined. RoleX.wsdl contains all portTypes, bindings and service definitions required for inter-process communication, while RoleX_backend.wsdl contains components for communication that is triggered by the backend system. Furthermore, these WSDL-files contain all related partnerLinkTypes, bindings, service definitions and variable properties. Both import the common message WSDL file (common_msg_<BTA-NameID>.wsdl) generated from every BTA in the business collaboration to be implemented. If a participating party never is the initiator of a BTA throughout a complete collaboration, the RoleX_backend.wsdl only contains the WSDL `definitions` tag without further content or document imports.

A BTA results in three different WSDL files. Two RoleX_c ommon_<BTA-NameID>.wsdl files that contain portType, binding, service definition, partnerLinkType and variable properties and import the aforementioned common_msg_<B TA-NameID>.wsdl containing common Types and Messages.

[8] `http://www.oasis-open.org/committees/ws-rx/`
[9] `http://www.oasis-open.org/committees/ws-tx`

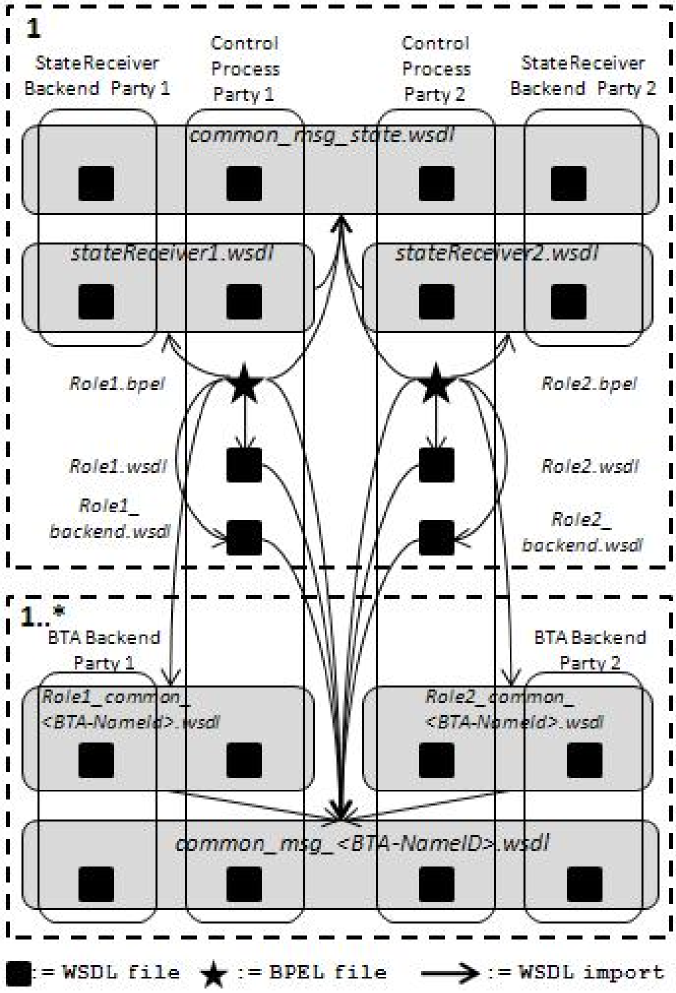

Figure 3: WSDL import relations

Exactly the same common_msg_<BTA-NameID>.wsdl file is distributed over the entire process to ensure seamless message routing while hiding system internal knowledge like endpoint references (in RoleX_common_<BTA-NameID>.w sdl) from the business partners. Together, RoleX_common_ <BTA-NameID>.wsdl and common_msg_<BTA-NameID> .wsdl form the interface for a role specific backend Web Service (per BTA), indicated by vertical bars without filling.

Altogether, each BPEL process imports common_msg_sta te.wsdl as well as the party specific WSDL files stateReceiverX.wsdl, RoleX.wsdl and RoleX_backend.wsdl. Moreover, the party specific WSDL interface generated for each BTA (RoleX_common_<BTA-NameID>.wsdl, common_msg _<BTA-NameID>.wsdl) is imported.

5. ebBP TO BPEL TRANSLATION

As pointed out in section 1, automatically deriving orchestration models from a common choreography definition is advantageous in terms of development speed and conformance. Therefore, this work presents a prototypic translator for the set of ebBP models defined in section 2.3. Such a translator supports conformance provided that it works "correctly". We claim that this is possible because validation of the translator is a one-time effort.

The translator covers the generation of BPEL processes, WSDL interfaces and deployment descriptors for the Apache ODE BPEL engine that has been used. This section concentrates on the most interesting part, i.e., the mapping of ebBP *BusinessCollaborations*, BTAs, *RequestingBusinessActivities* and *Decisions*. Note that BPEL elements not required for the understanding of the overall mapping concept are omitted.

Listing 2: BPEL process for a BusinessCollaboration

```
<process ... name="UseCase">
 <!-- WSDL imports here -->
 <!-- parterLinks here -->
 <!-- variables here -->
 <!-- correlation set here -->
 <scope name="UseCase">
  <eventHandlers>
   <onAlarm>
    <!-- collaboration timeout handling -->
   </onAlarm>
  </eventHandlers>
  <sequence>
   <!-- initialize process state -->
   <while>
    <condition>'true'</condition>
    <sequence>
    <!-- switch over states here -->
    <if>
    <!-- state AcceptableQuote reached -->
     <scope name="AcceptableQuote_Scope">
     <!-- declare request variables for permissible BTAs here -->
      <eventHandlers>
       <onAlarm>
        <!-- timeout handling for shared state AcceptableQuote-->
       </onAlarm>
      <eventHandlers>
      <while>
       <condition><!-- shared state not left --></condition>
        <sequence>
         <invoke operation="SIGNAL_BACKEND_CURRENT_STATE" ../>
         <pick>
          <!-- Permissible BTAs BPEL Code -->
         </pick>
        </sequence>
      </while>
     </scope>
    </if>
    <!-- ... switch over states ...-->
    </sequence>
   </while>
  </sequence>
 </scope>
</process>
```

The description starts out with the *BusinessCollaboration* construct that is mapped to a single BPEL `process` (per participant) that is depicted in simplified form in listing 2. Within the `process` several global "configurations" are specified such as WSDL imports and `partnerLink` definitions before a `scope` for the overall collaboration is defined. An `onAlarm` element is attached to this `scope` for capturing and handling timeouts defined for the overall collaboration. Further, a `sequence` is defined within this `scope` where process variables are initialized and a `while` construct is used that is executed until the process has terminated. For representing the collaboration's shared states, simple `if` elements are used within this `while` element to determine the shared state the collaboration is currently in and for each shared state another `scope` is defined that captures its "behavior". This `scope` also declares the variables required to save messages that can trigger the permissible BTAs of the shared state. Again, in case a timeout is defined for the shared state, another `onAlarm` construct is used to capture and handle the shared state's timeout. Then a `while` element is used to check whether the shared state has been left yet. If not, an `invoke` is used in order to notify the backend about the current process state which enables process stakeholders to be informed about the collaboration's progress. After that, a `pick` is used to wait for the backend or the integration partner's control process to trigger any BTA that is permissible for the respective shared state. The execution of such a BTA may then terminate the shared state's `while` using according variable assignments.

Table 1: BPEL production rules for ebBP BusinessTransactionActivity

Role	BPEL Process Elements
Initiator + Responder	- enclosing `onMessage`, receiving a triggering message from integration partner or backend system - enclosing `scope` for complete BTA - all `variables` required for the ebBP *Requesting-/RespondingBA* and the ebBP *Decision* - `catch` blocks for all ebBP failure types containing the corresponding reaction as specified in the ebBP *BusinessCollaboration* - `catchAll` block containing reaction as specified in ebBP *BusinessCollaboration* for *AnyProtocolFailure* - `onAlarm` to implement the *TimeToPerform* parameter specified for the BTA - `sequence` containing the BPEL code for the ebBP constructs (in this order): *RequestingBA*, *RespondingBA*, *Decision*. For the respective production rules see tables 2 and 3.

Having described the mapping of shared states, tables 1, 2 and 3 give a tabular overview of the most important BPEL elements used to translate a BTA with an attached Decision and indicate the purpose of their particular usage. The elements are listed in the order of their occurrence in the BPEL `process`. The depicted BTA contains a *ReceiptAcknowledgement* as well as an *AcceptanceAcknowledgement* signal. Further, all timing parameters offered for a BTA by the ebBP specification [12] are set. If only some or none of the signals for and parameters of a BTA are specified, the BPEL translation is a corresponding subset of the depicted one. As the mapping of a *RespondingBusinessActivity* is analogous to a *RequestingBusinessActivity*, only a *RequestingBusinessActivity* is described here. It is important to know that in BPEL, a `scope` in which a `fault` occurred is considered to have completed unsuccessfully [11]. Throwing a `fault` terminates all `scope`s this `fault` is thrown in or passed through until it is handled in some `scope`. Hence, an occurrence of an `onAlarm` based timeout in combination with throwing a `fault` terminates the BTA and is handled by the *faultHandlers* of the `scope` enclosing the BPEL code of a BTA. *ReceiptAcknowledgement/-Exception (RA/RAE)* and *AcceptanceAcknowledgement/-Exception (AA/AAE)* are processed concurrently as suggested by the ebBP specification

([12], sec. 3.4.9.3.3). A process tries to get a valid *RA/RAE* by sending the corresponding BD to the process of the integration partner until the specified ebBP *retryCount* is exceeded or a timeout occurs. Further, if both signal types are used, it waits to receive a valid *AA/AAE* until the occurrence of a timeout. At the end of the BTA mapping, an ebBP *Decision* is realized by using an `invoke` for querying the backend services for the evaluation of the latest business document exchanged and then a `process` global variable is set accordingly. This may lead to a switch to another shared state within the next iteration of the use case's `while` loop.

An ebBP *Start* element in combination with the first shared state it links to results in a variable assignment before the global process loop depicted in listing 2. Thereby, the global process variable is initialized with the value for the first shared state and as a result the permissible BTAs of this shared state are reachable. The translation of an ebBP final state *Success* or *Failure* results in an `invoke` statement to propagate the process state to the backend system and a subsequent `exit` command to terminate the process. Finally, the correlation of messages is described. As Web Services operate in stateless mode, messages have to be correlated using either WS-Addressing[10] or explicit addressing using BPEL `correlations`. Explicit correlation is used here as this option proved to be easier to implement for accessing backend services and is advantageous in terms of independence from the choice of BPEL engine. Clearly, this imposes the necessity on backend services to take over received correlators to response messages. Note that, nonetheless, many BPEL engines do support automatic correlation, especially for BPEL to BPEL communication.

6. PRACTICAL EXPERIENCE

The ebBP to BPEL translator has been written in the Java language and the main API used for that was the Streaming API for XML (StAX[11]). Approximately 14000 method lines of code have been written to implement the translator. Less code may have been needed using other libraries like DOM or other technologies like XSLT. For the approach presented here, the choice of technology for implementing the translator is of subordinate importance.

The use case of this work (section 3) can be translated in full and produces fully BPEL compliant process descriptions. The created BPEL processes have been tested using the Apache ODE 1.2 BPEL engine and the Apache Axis2 1.4 Web Service stack. For the backend services described above dummy Web Services have been implemented that emulate business logic by forwarding decisions to the user. Figure 4 shows the *Seller* role deciding whether to accept an order in full (Accepted) or to defer its decision (Pending). The use case from section 3 could not be performed on the selected platform in full due to an ODE bug in handling `whiles` in combination with `picks` which resulted in the case that a shared state's `while` element can only be entered once. Thus, though every shared state of the use case could be reached there were two states that could not be followed-on with a "normal" termination of the process. Furthermore, WS-ReliableMessaging (using Apache Sandesha2[12])

[10]`http://www.w3.org/TR/ws-addr-core`

[11]`http://jcp.org/en/jsr/detail?id=173`

[12]`http://ws.apache.org/sandesha/sandesha2/`

Table 2: BPEL production rules for ebBP RequestingBusinessActivity

Role	BPEL Process Elements
Initiator + Responder	- enclosing `scope` - enclosing `flow` for concurrent processing of *RA/RAE* and *AA/AAE*
	RA / RAE
Initiator	- enclosing `while` for trying to get a valid *RA/RAE* until ebBP *retryCount* is exceeded - `scope` to encapsulate *RA/RAE* handling - `catch` block for *RAE* handling - `invoke` to check *RAE* validity using the backend system - `throw` to throw ebBP *AnyProtocolFailure* in case no valid *RAE* was received and ebBP *retryCount* is exceeded - `throw` to throw ebBP *RequestReceiptFailure* in case of a valid *RAE* - `catchAll` block for technical failure (TF) handling - `rethrow` to forward TF to enclosing `scope` if ebBP *retryCount* is exceeded - `onAlarm` to implement ebBP *timeToAcknowledgeReceipt* - `throw` ebBP *AnyProtocolFailure* if ebBP *retryCount* is exceeded - `invoke` to forward Business Document (BD) to and get a *RA/RAE* from Responder - `invoke` to check *RA* validity using the backend system
Responder	- enclosing `scope` - `catch` block for *RAE* handling - `reply` construct to forward *RAE* to Initiator - `throw` ebBP *RequestReceiptFailure* in case of a *RAE* - `onAlarm` to implement ebBP *timeToAcknowledgeReceipt* - `invoke` to forward BD to and get a *RA/RAE* from backend system - `reply` to forward *RA* to Initiator
	AA / AAE
Initiator	- enclosing `while` to wait for valid *AA/AAE* - `scope` to encapsulate *AA/AAE* handling - `catch` block to forward ebBP *RequestAcceptanceFailure* faults to enclosing `scopes` - `catchAll` block to handle TF - `empty` to wait for an *AA/AAE* despite of TFs - `onAlarm` to implement ebBP *timeToAcknowledgeAcceptance* - `pick` to receive either *AA* or *AAE* - `invoke` to check *AA/AAE* validity using the backend system - `throw` ebBP *RequestAcceptanceFailure* in case of a valid *AAE*
Responder	- enclosing `scope` - `catch` block for handling *AAE* - `invoke` to forward *AAE* to Initiator - `throw` ebBP *RequestAcceptanceFailure* in case of an *AAE* - `onAlarm` to implement ebBP *timeToAcknowledgeAcceptance* - `invoke` to get *AA/AAE* from backend system - `invoke` to forward *AA* to Initiator

Table 3: BPEL production rules for ebBP Decision

Role	BPEL Process Elements
Initiator + Responder	- `invoke` to send BD of *RespondingBA* to backend system in order to get an evaluation - if no *RespondingBA* exists, BD of *RequestingBA* is used - `if` and `assign` statements to determine and switch to next process state Note that *ConditionGuardValues* are evaluated before *DocumentEnvelopes*.

Figure 4: Seller deciding upon quote request

and WS-Security (using Apache Rampart[13]) have been considered for implementing QoS features. Though it was possible to offer BPEL processes as secure and reliable Web Services, invoking other Web Services from BPEL processes in a reliable and secure manner was not possible. So the application of these standards has been canceled. Some other QoS features can be implemented by offering utility services. For example, such utility services could be used to implement non-repudiation by signing and storing business documents. These utility services could then be called from the control processes and these additional calls can quite easily be integrated into the translation engine as has been tested for a dummy non-repudiation service.

7. RELATED WORK

In general, this work is about implementing B2Bi using interacting partner processes. In so far, the work of standardization institutions that specify how to perform *BusinessTransactions* or similar concepts at runtime is related to our work. For example, the so-called RosettaNet Implementation Framework specifies rules which define how to perform PIPs. These standards, however, typically do not offer the generation of executable implementations of control processes as we do with our translator.

In the area of workflow research, the issue of using partner local processes for realizing B2Bi has been investigated. Issues like the conformance of local processes to global process descriptions have been analyzed, among others, in [18] from a conceptual point of view. The research problems investigated in that area are very much the same as in more recent contributions that explicitly analyze the dichotomy between *choreography* and *orchestration*, e.g., [21] propose *Let's Dance* as a language for modeling both, choreographies and orchestrations. These more conceptual approaches differ from our work in not using dedicated B2Bi standards like ebBP and BPEL and in frequently only covering the functionality of these standards partly, most notably message and control flow.

Several more technology driven approaches like [14] and [5] derive BPEL from micro-choreography compositions but do not offer a B2Bi standards based choreography model for the composition of micro-choreographies.

One goal of our work is to provide compatibility of interacting BPEL processes by deriving BPEL processes from common ebBP choreographies. While this is a constructive approach, the problem may be tackled in an analytical way as well, e.g., [9] analyze compatibility by means of defining a BPEL semantics in terms of Petri nets. Related to this are approaches like [20] and [1] that focus on analyzing conformance of orchestration models to choreography models in an analytical way. Note that more general findings from workflow research in general (cf. [18]) apply as well.

There are several contributions that translate dedicated B2Bi or Web service choreography languages to BPEL as we do. [7] also propose the translation of ebXML BPSS to BPEL. Apart from being designed for BPSS 1.1, [7] is different from our work in not applying a shared state based modeling approach to ebBP and in not reporting on a fully working translator.

An interesting approach is presented in [3] that proposes an UML profile for modeling the orchestration of multiple UMM *BusinessTransactions* of one integration partner. Such a model can then be transformed to BPEL processes. This differs from the work at hand in actually transforming a *UML orchestration* into a BPEL orchestration, but not that the *UML orchestration* is derived from one (or more) UMM choreographies which represents an extra step in deriving BPEL processes from choreographies that can be used for adding more partner-specific logic. Interestingly, [3] also captures collaboration *state* for routing the choreography but incorporates it in the model by using transition guards. Last, [3] assumes UMM and consequently UML for modeling choreographies whereas we only expect the textual format ebBP which may be more accessible.

Finally, there are several proposals for mapping WS-CDL choreographies to BPEL, e.g., [10] and [19]. These approaches differ from ours in using WS-CDL instead of ebBP. While WS-CDL may be a good choice for many choreographies due to its tight relationship with WSDL as opposed to ebBP, we claim that ebBP is particularly useful for B2Bi due to its better support for specifying QoS features.

8. CONCLUSION AND FUTURE WORK

The main goal of this paper, i.e., showing that *shared state* based ebBP models can be created in a standard compliant manner and subsequently translated into BPEL orchestrations has been achieved by describing a suitable modeling concept and by implementing a prototypic ebBP to BPEL translator. *Shared states* support the agreement function of choreography models and allow for the definition of state specific timeouts. They are beneficial for creating choreography as well as orchestration models by offering natural synchronization points and, finally, define the basis for signaling the collaboration's progress to process stakeholders.

Apart from introducing *shared states* into ebBP to BPEL translations, an integration architecture for using the generated BPEL processes has been proposed which does not require BPEL processes to be edited after generation. Comparing the size of the use case to the NES standard processes

[13] `http://ws.apache.org/rampart/`

it can further be stated that even collaborations of real-world size can be processed. Tests using the BPEL engine Apache ODE showed that the generated BPEL processes can be executed to a large extent. Though BPEL engines and Web Service standards addressing QoS features have not yet reached their full potential, tests are promising that the approach proposed may be applicable for real world B2Bis in the future.

Nonetheless, there are limitations to the approach presented. Apart from multi-party integrations and hierarchical decomposition for more complex models, support for QoS features like reliability or security is a key issue for the approach to be really useful in the B2Bi area. Furthermore, a process model for applying the approach within B2Bi projects should be defined, in particular when it comes to handling changes in the choreography. Focusing on the ebBP model an extension to the standard is envisaged that allows for a smoother integration of shared states, but we decided to check its expressibility in terms of regular ebBP constructs first. Currently, the Fork-Join work-around and the implementation of corresponding *Joins* assume a 1-to-1 relation between *Forks* and corresponding *Joins*. It also leads to some inelegant, yet standard compliant, constructs such as *Joins* referencing the same BTA in its *FromLinks*.

As the practical findings of this work are encouraging the use of ebBP to BPEL translations more formal analysis and validation features should be applied. In particular, the soundness of ebBP input models, the conformance of BPEL orchestrations to ebBP choreographies and the compatibility between interacting BPEL processes are to be investigated. As ebBP and BPEL both do not have formal semantics the development/selection of suitable semantics is the next step of our work.

9. REFERENCES

[1] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying the conformance of web services to global interaction protocols: A first step. In *EPEW/WS-FM, ser. Lecture Notes in Computer Science*, pages 257–271, 2005.

[2] A. Barros, M. Dumas, and A. H. M. T. Hofstede. Service interaction patterns. In *Proceedings of the 3rd International Conference on Business Process Management (BPM), Nancy, France*, pages 302–318. Springer Verlag, 2005.

[3] 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 and EEE'08*, Washington D.C., USA, July 2008. IEEE.

[4] J. T. Mentzer et al. Defining Supply Chain Management. *JOURNAL OF BUSINESS LOGISTICS*, 22(2):1–26, 2001.

[5] R. Khalaf. From RosettaNet PIPs to BPEL processes: A three level approach for business protocols. *Data Knowlegde Engineering*, 61(1):23–38, 2007.

[6] B. Kiepuszewski, A. H. Hofstede, and C. J. Bussler. On structured workflow modelling. *Lecture Notes in Computer Science*, 1789:431–445, 2000.

[7] J.-H. Kim and C. Huemer. From an ebXML BPSS choreography to a BPEL-based implementation. *SIGecom Exch.*, 5(2):1–11, 2004.

[8] D. M. Lambert and M. C. Cooper. Issues in Supply Chain Management. *Industrial Marketing Management*, 29(1):65 – 83, 2000.

[9] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting ws-bpel processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.

[10] J. Mendling and M. Hafner. From WS-CDL choreography to BPEL process orchestration. *Enterprise Information Management (JEIM). Special Issue on MIOS Best Papers*, 2006.

[11] OASIS Open. Web services business process execution language (wsbpel).

[12] OASIS Open. *ebXML Business Process Specification Schema Technical Specification.* OASIS Open, 2.0.4 edition, December 2006.

[13] OASIS Open. *Web Services Business Process Execution Language*, 2.0 edition, April 2007.

[14] A. Schönberger and G. Wirtz. Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study. In *The 2006 Int. Conf. on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing (CSREA EEE'06)*, June 2006.

[15] A. Schönberger and G. Wirtz. Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions. In *The 2006 Int. Conf. on Software Engineering Research and Practice (SERP'06)*, June 2006.

[16] 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.

[17] C. Schroth, T. Janner, and V. Hoyer. Strategies for cross-organizational service composition. In *MCETECH '08: Proceedings of the 2008 International MCETECH Conference on e-Technologies*, pages 93–103, Washington, DC, USA, 2008. IEEE Computer Society.

[18] W. M. P. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 140–156, London, UK, 2001. Springer-Verlag.

[19] I. Weber, J. Haller, and J. A. Mülle. Automated derivation of executable business processes from choreograpies in virtual organizations. In *Proceedings of Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006)*, Mar. 2006.

[20] W. L. Yeung. A formal basis for cross-checking ebxml bpss choreography and web service orchestration. In *APSCC '08: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, pages 524–529, Washington, DC, USA, 2008. IEEE Computer Society.

[21] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Let's dance: A language for service behavior modeling. In *Proceedings of the 14th international conference on cooperative information systems (CoopIS'06)*, pages 145–162, Montpellier, France, 10 2006.