COMPUTING SCIENCE

Designing Atomic Business Functions

Santosh Shrivastava and Mark Little

TECHNICAL REPORT SERIES

No. CS-TR-1457 March 2015

TECHNICAL REPORT SERIES

No. CS-TR-1457 March, 2015

Designing Atomic Business Functions

S. Shrivastava and M. Little

Abstract

Business-to-business integration (B2Bi) solutions offered by vendors fall into two broad categories: hub-and-spoke (interaction between partners takes place through a central hub that acts as an intermediary), and peer-to-peer (interaction takes place directly between partners). Vendors are increasingly offering cloud based solutions; a hub-and-spoke architecture is well suited to SaaS level provision whereas a peer-to-peer architecture offers a wider set of options, allowing individual partners to deploy their side of software at the level of IaaS, PaaS or SaaS. An important coordination problem in B2Bi that needs addressing is how to ensure that business interactions between partners terminate in a consistent manner even in the presence of application level exceptions and software, hardware and network related problems commonly encountered in distributed systems. Solutions that have been developed so far and incorporated in SOA middleware are essentially based on OASIS WS-TX set of transaction standards, namely WS-coordination, WS-AtomicTransaction and WS-BusinessActivity. WS-TX based solutions require a central activity coordinator. The paper argues that although these solutions are quite suitable within a hub- and spoke B2Bi architecture, they sit awkwardly in peer-to-peer B2Bi settings, where a distributed approach, not requiring a central coordinator is more suitable. The paper develops such an approach; it focuses on the choreography of the business function and describes how to make the choreography atomic, ensuring consistent termination in the presence of application level exceptions and failures.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

SHRIVASTAVA, S., LITTLE M.
Designing Atomic Business Functions
[By] S. Shrivastava, M. Little
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1457)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1457

Abstract

Business-to-business integration (B2Bi) solutions offered by vendors fall into two broad categories: hub-and-spoke (interaction between partners takes place through a central hub that acts as an intermediary), and peer-to-peer (interaction takes place directly between partners). Vendors are increasingly offering cloud based solutions; a hub-and-spoke architecture is well suited to SaaS level provision whereas a peer-to-peer architecture offers a wider set of options, allowing individual partners to deploy their side of software at the level of IaaS, PaaS or SaaS. An important coordination problem in B2Bi that needs addressing is how to ensure that business interactions between partners terminate in a consistent manner even in the presence of application level exceptions and software, hardware and network related problems commonly encountered in distributed systems. Solutions that have been developed so far and incorporated in SOA middleware are essentially based on OASIS WS-TX set of transaction standards, namely WS-coordination, WS-AtomicTransaction and WS-BusinessActivity. WS-TX based solutions require a central activity coordinator. The paper argues that although these solutions are quite suitable within a hub- and spoke B2Bi architecture, they sit awkwardly in peer-to-peer B2Bi settings, where a distributed approach, not requiring a central coordinator is more suitable. The paper develops such an approach; it focuses on the choreography of the business function and describes how to make the choreography atomic, ensuring consistent termination in the presence of application level exceptions and failures.

About the authors

Santosh Shrivastava was appointed Professor of Computing Science, University of Newcastle upon Tyne in 1986 where he is now an Emeritus professor. He received his Ph.D. in computer science from Cambridge University in 1975. The emphasis of his work has been on the development of concepts, tools and techniques for constructing distributed fault-tolerant systems that make use of standard, commodity hardware and software components. Current focus of his work is on middleware for supporting inter-organization services where issues of trust, security, fault tolerance and ensuring compliance to service contracts are of great importance as are the problems posed by scalability, service composition, orchestration and performance evaluation in highly dynamic settings. He is a member of IFIP WG6.11 on Communication Aspects of the E-World, and sits on the advisory board of Arjuna technologies Ltd which he co-founded in 1998.

Dr Mark Little works for Red Hat where he is VP of Engineering and a Red Hat Fellow, responsible for JBOSS Technical Direction as well as Research and Development. Following a PhD at Newcastle University, he became Chief Architect and co-founder at Arjuna Technologies. He has been working in the area of reliable distributed systems since the 1980s-his PhD was on fault-tolerant distributed systems, replication and transactions. He holds a position as visiting professor at Newcastle University.

Suggested keywords

BUSINESS-TO-BUSINESS INTEGRATION
EXTENDED TRANSACTIONS
CHOREOGRAPHY
ORCHESTRATION
RELIABLE MESSAGING

Designing Atomic Business Functions

Santosh Shrivastava
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
Email: santosh.shrivastava@ncl.ac.uk

Mark Little
Red Hat Ltd.
The Core, Science Central,
Newcastle upon Tyne, UK
Email: mlittle@redhat.com

***Abstract*—Business-to-business integration (B2Bi) solutions offered by vendors fall into two broad categories: *hub-and-spoke* (interaction between partners takes place through a central hub that acts as an intermediary), and *peer-to-peer* (interaction takes place directly between partners). Vendors are increasingly offering cloud based solutions; a hub-and-spoke architecture is well suited to SaaS level provision whereas a peer-to-peer architecture offers a wider set of options, allowing individual partners to deploy their side of software at the level of IaaS, PaaS or SaaS. An important coordination problem in B2Bi that needs addressing is how to ensure that business interactions between partners terminate in a consistent manner even in the presence of application level exceptions and software, hardware and network related problems commonly encountered in distributed systems. Solutions that have been developed so far and incorporated in SOA middleware are essentially based on OASIS WS-TX set of transaction standards, namely WS-coordination, WS-AtomicTransaction and WS-BusinessActivity. WS-TX based solutions require a central activity coordinator. The paper argues that although these solutions are quite suitable within a hub-and spoke B2Bi architecture, they sit awkwardly in peer-to-peer B2Bi settings, where a distributed approach, not requiring a central coordinator is more suitable. The paper develops such an approach; it focuses on the choreography of the business function and describes how to make the choreography atomic, ensuring consistent termination in the presence of application level exceptions and failures.**

Keywords*—*Business-to-business integration, Extended transactions, Choreography, Orchestration, Reliable messaging

I. Introduction

Our domain of interest is business-to-business integration (B2Bi) that autonomous organizations - business partners - need to do in order to automate their *business functions* such as travel booking, order fulfilment and so forth. B2Bi software solutions offered by vendors fall into two broad categories: *hub-and-spoke* (interaction between partners takes place through a central hub that acts as an intermediary), and *peer-to-peer* (interaction takes place directly between partners). Vendors are increasingly offering cloud based solutions; a hub-and-spoke architecture is well suited to SaaS level provision whereas peer-to-peer architecture offers a wider set of options, allowing individual partners to deploy their side of software at the level of IaaS, PaaS or SaaS.

The interaction part of a given business function is composed of primitive operations, where each such operation involves sending of a message representing a single electronic business document that contains either a request for carrying out a specific, well defined function by the recipient party (e.g., "verify that a customer credit card is valid and can be used as a form of payment for the amount requested", "request purchase order cancellation") or a notification of an action performed by the sending party (e.g., "invoice notification", "goods dispatched"). We note that messaging facilities are typically implemented using message oriented middleware (MoM) that offers reliable, persistent messaging with well defined operations (e.g., connect, disconnect, deposit message, pickup a message) that enable interactions between loosely coupled parties (in that they are not always required to be 'on-line at the same time').

At a higher level, the message-based interactions can be viewed as the business partners taking part in the execution of a shared business process (also called *public* or *cross–organizational* business process), where each partner is responsible for performing their part in the process. Naturally, executions at each partner must be coordinated reliably at run-time to ensure that the partners are performing mutually consistent actions (e.g., the seller is not shipping a product when the corresponding order has been cancelled by the buyer). In effect, an important problem that needs addressing is how to ensure that business interactions terminate in a consistent manner even in the presence of application level exceptions and software, hardware and network related problems commonly encountered in distributed systems (e.g., clock skews, unpredictable transmission delays, message loss, node crashes etc.).

This is a very topical subject that has received much attention, particularly within the service oriented architecture (SOA) computing community. Solutions that have been developed so far and incorporated in SOA middleware are essentially based on database centric distributed ACID transactions and their non-ACID counterparts for long running activities, the best known one being the OASIS Web Services set of Transaction standards, namely, WS-coordination, WS-AtomicTransaction and WS-BusinessActivity (collectively referred to as WS-TX [1]). WS-TX based solutions require a central 'activity' coordinator and fit well within hub-and spoke B2Bi architecture where the hub is the natural place for the home of the coordinator. Unfortunately, centralized coordinator based solutions sit awkwardly in peer-to-peer B2Bi settings as it is frequently not obvious who should 'own' the coordinator. Another reason is the reluctance of application builders to deploy protocols (e.g., for commit processing) that require application-level coordination across multiple entities within or across organizations, as these protocols are seen to impinge upon application scalability and autonomy [2]. It is therefore worth investigating an alternative, a solution that avoids the

need for a central coordinator and at the same time has minimum impact on organizational autonomy. Such a solution would then fit well with peer-to-peer B2Bi architectures and therefore offer flexible cloud deployment options.

In this paper we propose one such alternative: the main contribution of the paper. Our approach offers broadly similar functionality as WS-BusinessActivity, but sidesteps the database centric view of consistency and focuses instead on the *choreography*: for a given business function, its choreography specification is a description, from a global perspective, of all the permissible message exchange sequences between the partners. We develop a simple notion of what it means for given sequence to be consistent, and state that a business function is atomic if all the execution sequences of its choreography are consistent. Equivalently, a choreography is atomic if all of its execution sequences are consistent. We propose a systematic way of developing atomic choreographies that takes into account application level exceptions and failures. Our approach is amenable to automated correctness analysis using well-known model checking techniques. We show that atomic choreographies are directly amenable to deriving distributed coordination in that there is no need for an activity coordinator (that would be required in a WS-TX based solution) for directing confirmations, cancellations, compensations etc.

The paper is structured as follows: related work is presented in section two; section three presents the basic concepts that are then used in section four for the development of atomic choreographies, illustrated with two hypothetical examples of the type that are often used for illustrating the workings of WS-BusinessActivity. Section five presents our ideas on tool support required for developing atomic choreographies. Concluding remarks are presented in section six.

II. Related work

See [3] for a good discussion architectural issues in B2B integration. In [4] the authors describe the design of a cloud based messaging gateway that is in style of peer-to-peer B2Bi. This paper [5] contains a good discussion on challenges facing designers of service-oriented applications in ensuring that "the autonomous services that make up these distributed applications always finish in consistent states". Many advanced transaction models have been proposed in the literature, a good survey is presented in [6]. The OASIS WS-TX set of ACID and non-ACID transaction standards [1] has been widely adopted in industry. These standards make use of the concept of an activity service originally proposed for CORBA middleware [7]. Using WS-TX as the basis, various researchers, e.g., [8],[9], have proposed advance transaction management frameworks for SOA.

In a widely cited paper, Helland, a transaction architect from industry, observes that in practice, message based applications rarely make use of distributed transactions, instead, "applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses" [2]. Helland and others [10] go on to state that 'performing tentative business operations' is a way that gets around not having distributed transactions. Thus a party will send a message that requests an operation to be performed tentatively, leaving open the possibility of cancellation: "Essential to a tentative operation, is the right to cancel. Sometimes, the entity that requested the tentative operation decides it is not going to proceed forward. That is a cancelling operation. When the right to cancel is released, that is a confirming operation. Every tentative operation eventually confirms or cancels." [2]. These observations have motivated our approach.

This [11] is one of the earliest papers that explored atomicity within the context of e-commerce. Tygar defined *Goods-atomic* protocols that "effect an exact transfer of goods for money". Our concept of atomic business function can be seen as a generalization of this idea. Atomic business functions require choreographies that are 'contract compliant'. To this end we have used our earlier work on contract compliance and contract-choreography conformance [12], [13]. One of the earliest work that explored connection between contracts and choreographies is [14]; other relevant work is reported in [15] and [16], [17].

OASIS ebxml and RosettaNet are influential industry standards [18], [19], that stressed the importance of state alignment in B2B interactions. Other researchers have investigated performing state synchronization at application level [20]. The approach of using a choreography specification for deriving the individual business process components of partners is gaining wide attention, see for example [21]. SAVARA is a good example of an industrial framework for choreography design [22]. It provides tools for designing choreographies using BPMN 2.0 notation and deriving partner business processes (expressed in BPEL). The RosettaNet way of designing choreographies [23], [24], uses single message business operations with normal and exceptional outcomes with state-alignment, just as we do, and illustrates the practicality of our approach.

III. Basic concepts

We take a closer look at B2B message-based interactions and observe that more often than not, the parties are in a peer-to-peer relationship as against the more traditional client-server relationship. Fig. 1(a) shows a typical interaction between a buyer, a seller and a shipper (Fig. 1(b) indicates a variation on that interaction that we will discuss shortly).

The buyer sends a message (PO_Req) requesting processing of a purchase order for some specific goods to the seller; if the seller has the requested goods in stock and can arrange shipping (Shipping_Req, Shipping_Confirm interaction with the shipper), it sends PO_Confirm to the buyer (otherwise it is supposed to send PO_Reject message), and so on. In such interactions, any peer can initiate the transfer of a message at any time, so there is always a danger of race conditions occurring. For example, after having sent PO_Req, the buyer might request a cancellation by sending a PO_Cancel message, just as the seller sends a PO_Confirm message. What happens next will be determined by the details of the business agreement regarding order cancellation that exists between the buyer and the seller (assuming of course, that the order cancellation part of the cross–organizational business process has been correctly designed and implemented according to this agreement). An inconsistent final state in which the buyer assumes that order cancellation has taken place while the seller is awaiting payment must be avoided.

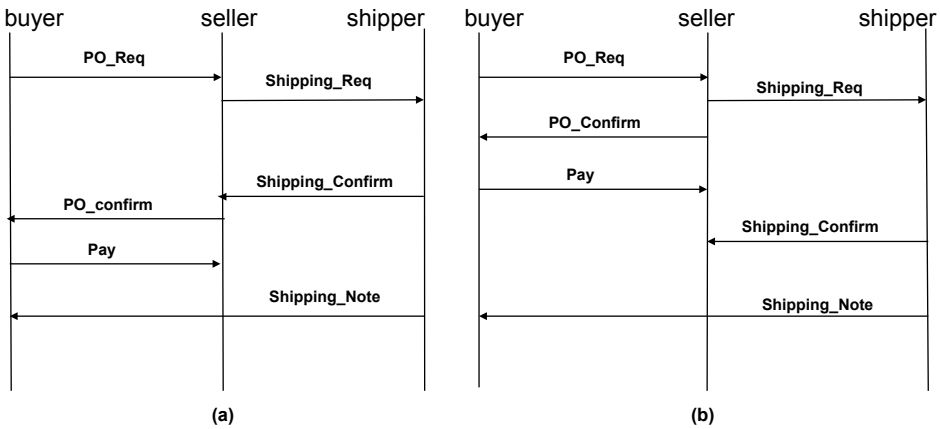

Fig. 1. Buyer-seller-shipper interactions.

Given loose coupling between partners, a common interaction pattern encoded in a business function consists of parties making progress based on *tentative commitments* (tentative promises) made to each other, with each party prepared to deal with exceptions if changes occur, in a manner that is consistent with the business agreements between the parties [2]. For example, the seller might have a flexible agreement with the buyer that enables the seller to deal with orders promptly, by sending PO_Confirm and accept payment before shipping confirmation has occurred, Fig. 1(b). In (hopefully rare) cases where timely shipping cannot be performed (the shipper sends Shipping_Reject), the agreement permits the buyer to make a refund or request delayed shipping.

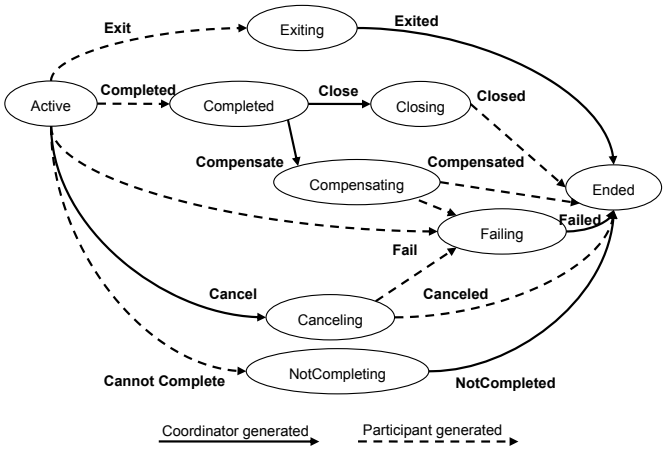

Fig. 2. BusinessAgreementWithParticipantCompletion state diagram

As remarked earlier, the best-known approach to ensuring that business interactions terminate in a consistent manner involves managing a given interaction as a non-ACID transaction, WS-BusinessActivity. Essentially, in such an approach, the responsibility of maintaining consistency is shared between the participants and an activity coordinator in the following manner: the participants are required to 'register' themselves with the coordinator before starting their individual computations (activities); they subsequently inform the outcomes of their activities (completed, fail, etc.) to the coordinator. The coordinator then directs them to individually close the computation (= commit) or compensate. Fig. 2 shows the coordination messages that can be generated for the BusinessAgreementWithParticipantCompletion variant of the WS-BusinessActivity protocol, which assumes that a participant knows when it has completed all work for a business activity, and informs the coordinator. The expected sequence is that an activity reaches the completed state then it enters the final ended state after successfully compensating ('compensated') or closing ('closed') as directed by the coordinator.

One must admit the possibility of an activity failing to compensate/close/cancel (= abort): this results in the failed notification from the coordinator, indicating that the business activity has terminated in an inconsistent state. Any recovery measures undertaken would be outside the framework of WS-BusinessActivity. In a well designed system, such an occurrence would be quite rare, and would probably be related to unrecoverable problems at the infrastructure level.

In any solution that does not make use of a coordinator, each participant needs to have enough state information about the state of the overall computation to enable it to make the choices that would have been made by the coordinator on behalf of the participant. This implies that the participants will have to perform some coordination related message exchanges between themselves.

With the above discussion in mind, we next describe our overall approach. The idea is to tie the notion of consistency to the choreography since message exchanges between partners is the only observable behaviour. To begin with, we assume business functions involving just two parties. For a given business function, our aim is to come up with a choreography specification that accurately captures all permissible interactions that the business agreement between the partners allows. This can be a difficult task, so we would be looking for some tool support that assists the development of a choreography (see section V). The choreography specification needs to be 'executable', to enable deriving individual business process components of the partners; the specification should also take into account various exceptions and failures that can occur.

A choreography and a *business agreement* (a *business contract* or *contract* for short) are two specifications that describe permissible interactions between partners from different view points, emphasising different aspects. Whereas a choreography

specification is a description, from a global perspective, of all permissible message exchange sequences between the partners, a contract specifies what operations the business partners have the rights, obligations or prohibitions to execute; it also stipulates when the operations are to be executed and in which order. With some simplifications, the buyer and the seller, Fig. 1, might have a contract along the following lines:

1) *The buyer can place a purchase order request with the seller.*
2) *The seller is obliged to respond with either confirmation or rejection within 2 days of receiving the request.*
    a) *No response from the seller within 2 days will be treated as a rejection.*
3) *The buyer can either pay or cancel the purchase request within 3 days of receiving a confirmation.*
    a) *No response from the buyer within 3 days will be treated as a cancellation.*

Choreography and contract specifications are related to each in the following intuitive manner: at the beginning, partners have certain rights and prohibitions; as an interaction progresses, new rights, obligations and prohibitions come in force and and some old ones get revoked. For example, after the buyer has exercised the right to place a purchase order, an obligation on the seller to respond appropriately comes in force.

It is naturally important to make sure that message exchange sequences of a given choreography conform to (are in accordance with) the contract between the partners. In other words, make sure that all message exchange sequences of a choreography are *contract-compliant*. We say that a given choreography execution sequence is *consistent* if in the terminated state there are no pending obligations (implying all the partners have performed their duties). By definition, all contract-compliant sequences are consistent. A choreography (business function) is *atomic* if all of its execution sequences are consistent.

A business function involving more than two parties can have more than one contract, each involving just a pair of partners. In the example of Fig. 1, two contracts are involved, buyer-seller and seller-shipper. There is no contract between buyer-shipper (the buyer is just receiving a notification from the shipper who has an obligation to do so under the seller-shipper contract). Thus, an atomic choreography involving more than two partners can be knit together from two party atomic choreographies.

As stated in the introduction, a business function is built from primitive business operations, where each operation is between a pair of parties and involves sending of just a single business message. Associated with the message sending operation is a little bit of handshake synchronization for 'state alignment' that ensures that both the parties have a common understanding of the status of the operation termination (whether the message has been delivered *and* accepted by the receiver for processing). We explain this further in the next section. This is the basic coordination mechanism we need for composing executable atomic choreographies.

IV. Atomic Choreographies

A. Business operations and state alignment

We assume that business functions are designed according to the general pattern of making progress based on tentative commitments as discussed earlier. Thus the set of operations (business messages) belonging to a business function will be of type 'request', 'confirm', 'cancel'. See the industry standard specifications from specific business domains, such as RosettaNet (supply chain management) [19] and Opentravel Alliance (hotel, travel industry) [25] for the set of business messages following this pattern.

Taking the cue from the RosettaNet [19] and ebXML [18], we note that sending of a business message is supported by a fairly sophisticated messaging protocol, as business messages usually have timing and validity constraints: a received document is accepted for processing by the receiver only if the document is received within the set time-out period (if applicable) and the document satisfies a number of security, syntactic and semantic validity checks (else the document is rejected). Thus, the messaging protocol requires the receiver to send one or more 'signal' messages (also known as 'state alignment' messages) to the sender indicating whether the validity checks have succeeded or not. The idea is to make sure that the public business processes at the sender and at the receiver have identical information regarding the fate of the business message: whether the receiver has accepted the document for processing ('success') or not ('failure' with perhaps some additional information indicating type of failure). State alignment is necessary to make sure that public business processes at each end make mutually consistent progress. Of course, signal messages could themselves get lost, or arrive after the time out set by the sender expires, in which case there is still a danger that the states of the sender and the receiver might not be aligned. An *outcome synchronization* mechanism is required to prevent this from happening (see [26], [27]). In particular, outcome synchronization ensures that if any party raises a failure exception, then the send operation will return fail to both the parties.

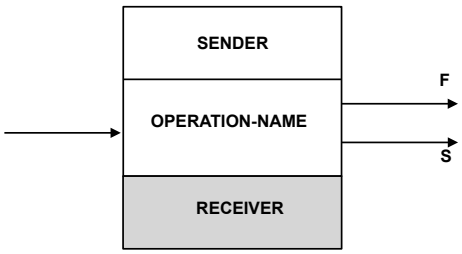

Fig. 3. BPMN task notation

Outcome synchronization between business partners is performed at the level of middleware and not at application level, so need not impact on organizational autonomy. For example, [26] shows that synchronization can be performed by using the same three-way handshake protocol which is widely used to manage TCP connections [28].

We abstract away these details, and assume that once a business operation is initiated it always completes to produce an outcome result of the operation that is known to both the

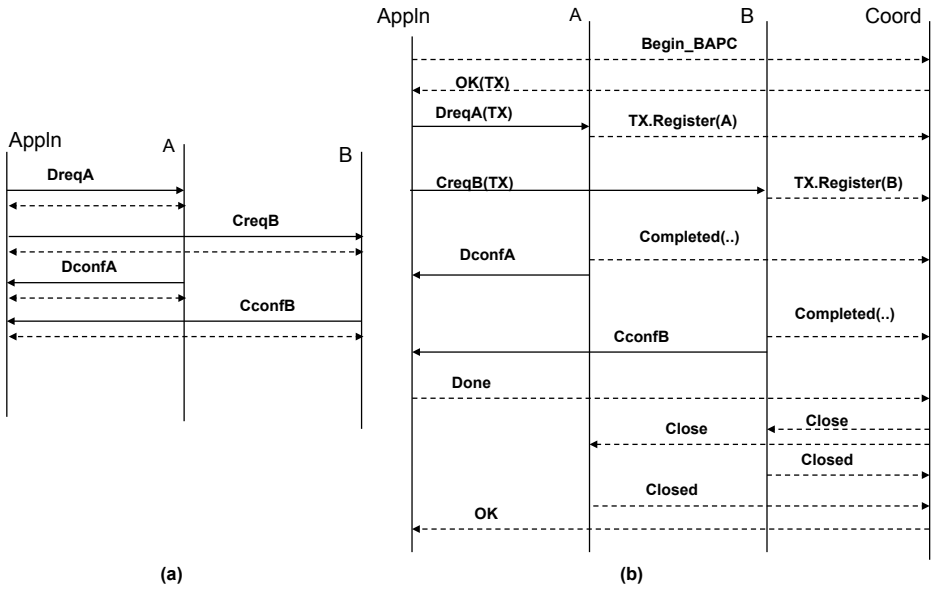

Fig. 4. Money transfer.

parties. This outcome is *successful* (*S*), indicating that the sent business document has been accepted for processing by the receiving partner or *fail* (*F*), indicating that the document has not been accepted for processing at the receiver.

The fail outcome can contain some additional details indicating the cause of the failure. There are three broad categories of failures: *initiation failure*, indicating a failure to establish a connection with the receiver, *technical failure*, indicating a problem at the protocol level, such as a late, syntactically incorrect or a missing message, and *business failure*, indicating some semantic error in the message, such as invalid purchase order number. The failure outcome corresponds to the failed notification in WS-BusinessActivity (see Fig. 2). In a well designed system, messages would be expected to be semantically correct, so failure handling by the sender in the form of a finite number of retries should normally lead to a successful outcome; repeated failure occurrences would be quite rare, and would probably be related to unrecoverable problems at the infrastructure level. At this stage, 'out of band' communication between partners would be needed to terminate the whole interaction. Ideally, at the contractual level, there should be a contract clause acknowledging such a possibility along the lines:

"Failure handling: if even after repeated attempts, an operation does not succeed, then the contractual interaction shall be declared terminated."

We make use of BPMN 2.0 notation [29] for depicting choreographies. Fig.3 shows the general form of a BPMN task (with two mutually exclusive outcomes) that represents a business operation.

B. Example: money transfer

Our first example concerns transferring some money from an account at bank A to an account at bank B. Although this example is frequently used in textbooks to illustrate why one needs distributed transactions, in reality, banks avoid using a distributed transaction solution as it would require one bank holding locks on the database of another bank. Rather, they settle for using local atomic operations strung together with messages and offer a tentative promise to the requester (along the lines: "the amount will usually be credited to the beneficiary's account within 2 hours, subject to normal fraud checks.").

We assume twelve business operations (six for each bank): DreqA (request bank A to debit a given amount from an account at A), CreqA (request bank A to credit a given amount from an account at A), DconfA (requested debit operation at A confirmed), DrejA (requested debit operation at A rejected), CconfA (requested credit operation at A confirmed) and CrejA (requested credit operation at A rejected). There is a complementary set of six operations at bank B. As stated earlier, we assume that each operation generates either a success outcome or a failure outcome, indicated using a superscript, e.g., $DreqA^s, DreqA^f$. Usually, there will be an autonomous service at each bank that offers a transfer service to its customers (Appln belonging to A in this example), and the two banks will have an agreement that states that credit/debit requests will be honoured promptly with confirmation or rejection (giving reason).

Fig. 4 (a) depicts a message sequence when everything succeeds (each operation generates a success outcome, and a requested operation is later confirmed). We use a dotted double arrowed line to indicate state alignment (which is an integral part of a business operation and performed automatically), and a solid arrowed line to indicate the business message that the corresponding business operation sends. The business logic is straightforward: an application (Appln) sends debit and credit requests to banks and awaits responses; A (B) performs the operation and sends confirm. The (b) part of this and the next figure depicts coordinator based solution that will be discussed shortly.

A situation requiring compensation is depicted in Fig. 5

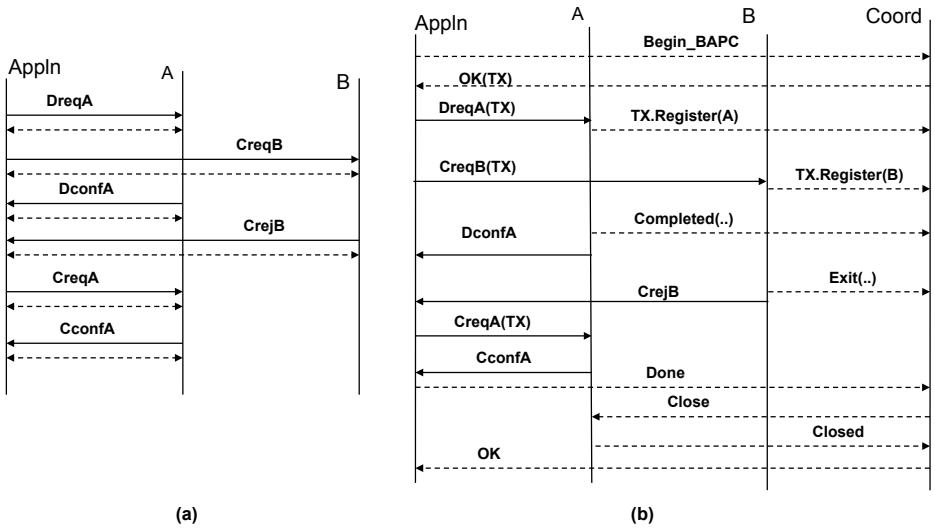

Fig. 5. Money transfer: compensation.

(a): here B (for whatever reason) rejects the credit request, in which case, Appln is obliged to credit the account at A, which it does.

The choreography is depicted in Fig. 6. We assume that the failure outcome of an operation results in the choreography terminating in an abnormal state. In practice, the operation could be retried a few times before admitting defeat (this logic is not shown for the sake of simplicity). In the figure, G1 is a parallel gateway and G2 is a complex gateway. Responses are collected at G2 to decide which way the application should head. Path 2 depicts the situation of Fig. 5 (a). The choreography is relatively simple, so is not too difficult to check manually that successful terminations are atomic: a transfer succeeds (credit, debit requests are confirmed) or no transfer takes place (thanks to compensation if necessary).

Solutions using WS-BusinessActivity are depicted in the (b) parts of Fig. 4 and Fig. 5. Basic understanding of the activity protocol is assumed (reference [30] contains a good introduction). We make use of BusinessAgreementWithParticipantCompletion variant (BAPC for short), where a participant knows when it has completed all work for a business activity, and informs the coordinator. Here business messages are implemented as rpcs: the sender sends a request, the recipient performs validity checks and sends a response indicating whether the message has been accepted for processing or not (for the sake of simplicity, response messages are not shown). As before, coordination messages are shown using dotted lines. In Fig. 4 (b), Appln starts a new activity by sending Begin_BAPC and receives a transaction context (TX); a request such as DreqA contains this context information, so A can register itself as a participant with the coordinator (message TX.Register(A)). After informing the coordinator of completion (Completed messages), A and B send confirm messages to Appln. Note that the state alignment after each message as performed in the distributed version (Fig. 4 (a)) is not necessary here as the participants send the necessary information to the coordinator who builds the global picture. So, in the case requiring compensation (Fig. 5 ((b)), B sends an Exit message to the coordinator before sending reject message, CrejB, to Appln. The figure indicates Appln performing compensation; an alternative is also possible, whereby Appln sends cancel to the coordinator who informs A to compensate.

C. Example: Buyer-seller-shipper

Message sequences for the buyer-seller-shipper example, when there are no failures and requested operations are later confirmed are depicted in Fig. 7; both the versions are shown. It is assumed that Shipping_Note is a best effort message. In the coordinator version, seller-shipper interaction is performed as a nested activity (transaction context TX2).

Two choreographies, linked through the seller are involved; the buyer-seller contract (shown in section III) will form the basis of the corresponding choreography, which is shown in Fig 8. As before, we assume that a failed operation is retried a few times before admitting defeat; this logic is not shown for the sake of simplicity. In the absence of state alignment, application-level timeouts will be a potential source of state divergence between interacting parties. Consider this: the seller responds before its two day timeout expires, however the response arrives at the buyer after the buyer's timeout has expired. The buyer should reject the late message by raising a fail exception, and alignment will ensure fail status at the seller side as well. The buyer-seller contract will have a component concerned with refund processing, in case the seller is unable to arrange shipping after accepting payment. This aspect is not discussed here.

In summary, these examples illustrate the similarities and differences between the two approaches. We have the same business message interaction, but supported underneath by different forms of coordination. In the distributed version, state alignment is performed after message delivery to provide *bilateral* consistency guarantee: both the sender and the receiver need to have identical information regarding the fate of the business message. In the coordinator version, business messages delivered using rpcs (with the response indicating whether the message has been accepted for processing or not) are sufficient, as through some additional coordination

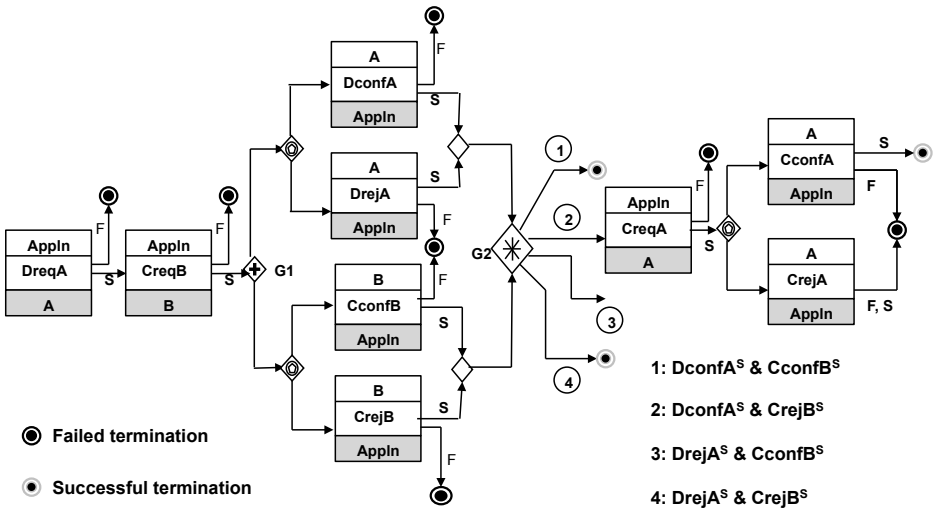

Fig. 6. Money transfer choreography.

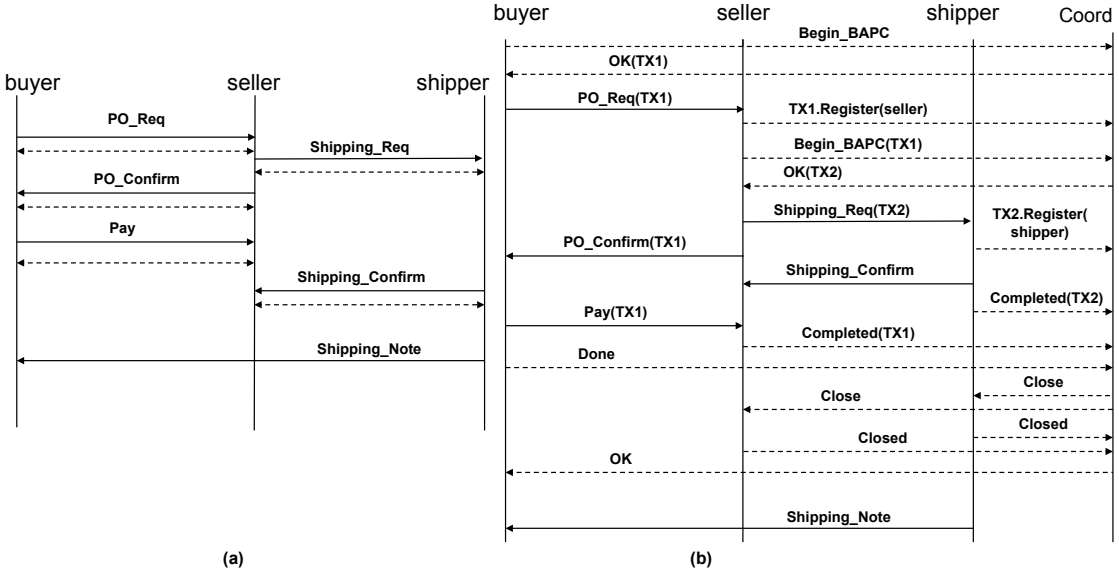

Fig. 7. Buyer-seller-shipper interactions.

messages, the coordinator collects the necessary information to get the global view.

V. Tool support

The fairly simple illustrative examples indicate that treatment of failures and timeouts introduce complexities in choreographies. For realistic examples, it would be quite difficult to establish manually that the choreographies are contract-compliant (therefore by definition, atomic); hence the desirability of having automated tool support. We have developed an approach to establishing conformance between a contract and a choreography that is described in detail in [13]. We have developed model checker based verification tools for contracts and BPMN choreographies that enable us to verify that execution sequences produced by a choreography are 'accepted' by the contract (and *vice versa*) [31],[32].

One of the challenges in developing such tools is that contract and choreography specifications describe permissible interactions between partners from different view points, emphasising different aspects. This is reflected in the fact that each has its own set of notations, design, verification and validation tools.

The conceptual framework that we have used for building model checker tools for contracts and choreographies is shown in Fig. 9 that depicts a contract monitoring system capable of observing B2B interactions and determining whether they are compliant with the contract. The figure shows a contract monitoring service called *Contract Compliance Checker (CCC)* that is deployed by the contractual parties (a buyer and a seller in this particular case) to monitor their B2B interactions at run time. The CCC is provided with an executable contract specification (derived from the natural language description of the business contract, as depicted by the dotted arrowed line) and observes significant messaging events, referred to here

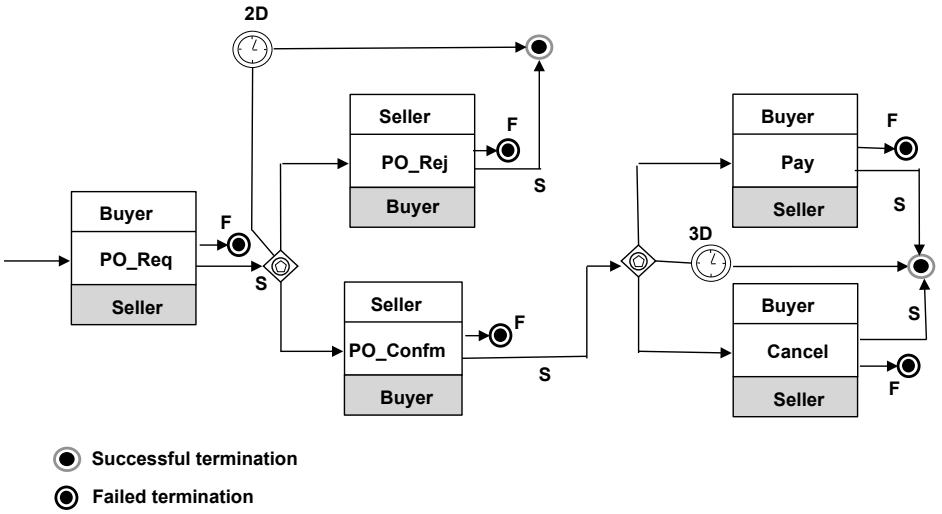

Fig. 8. Buyer-seller Choreography

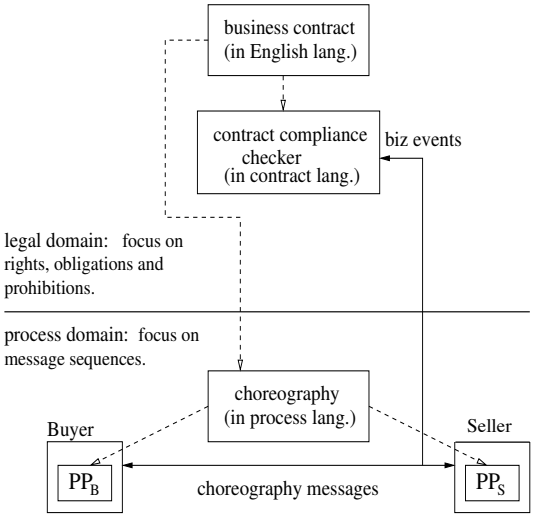

Fig. 9. Contract-choreography compliance.

as business events (*biz events*) produced from the interaction between the two parties and analyse them. The business contract also forms the basis for deriving a choreography specification. This specification in turn is used for deriving public processes of each partner (PP_b and PP_s of buyer and seller respectively).

The CCC and the choreography are modelled by Finite Automaton (FA) that accept languages over the same alphabet. Outcome events of business operations, (biz events, Fig. 9) such as bo_i^s, bo_i^f for operation bo_i form the common alphabet.

We have used the SPIN model checker [33]. Its input language for model building is PROMELA and the list of correctness properties that the model is expected to satisfy are expressed using Linear Temporal Logics (LTL). The verification includes two stages: independent verification and combined verification. Firstly, the two models are verified independently to guarantee that they satisfy certain correctness requirements specific to their domains. In the second stage, the behaviour of the previously verified models are contrasted against each other by means of comparison of their execution sequences. This is explained further in [13].

Conformance of choreography, but with focus on implementation, is also studied in [34]. In this work the implementation is produced automatically (by means of projection) from the choreography; consequently, the goal is to produce *realizable choreographies* that by definition will project conformant implementations. Like in our work, these authors use software tools (model checkers) for sequence generation and comparison. Another relevant work is reported in [35], where the authors describe a method of checking whether the orchestrations satisfy the temporal constraints of a choreography.

We are able to establish conformance between a contract and its choreography for a business function involving two parties. Additional research work is required to extend this to business functions involving more than two parties. Fortunately, most structures consist of composition of two party choreographies with well defined interaction paths between them (buyer-seller-shipper illustrates a simple case), so it should be possible to extend our approach.

VI. Concluding Remarks

In business-to-business message based interactions, executions at each collaborating partner must be coordinated reliably at run-time to ensure that the partners are performing mutually consistent actions. Solutions that have been developed so far and incorporated in SOA middleware are essentially based on database centric distributed ACID transactions and their non-ACID counterparts for long running activities, the best known one being the OASIS WS-TX set of transaction standards. We observed that WS-TX based solutions require a central activity coordinator; although these solutions are quite suitable within a hub-and spoke B2Bi architecture, they sit awkwardly in peer-to-peer B2Bi settings. This motivated us to look into an an alternative, a distributed solution that would fit well with peer-to-peer B2Bi architectures. As noted in the introduction, a peer-to-peer architecture offers a wider set of cloud deployment options than hub-and spoke, allowing individual partners

to deploy their side of software at the level of IaaS, PaaS or SaaS.

Our approach offers broadly similar functionality as WS-BusinessActivity, but sidesteps the database centric view of consistency and focuses instead on the choreography. We noted that given loose coupling between partners, a common interaction pattern encoded in a business function consists of parties making progress based on tentative commitments (tentative promises) made to each other, with each party prepared to deal with exceptions if changes occur, in a manner that is consistent with the business agreements (contracts) between the parties. We therefore developed the notion of consistency of choreography that is tied to the contract.

In any solution that does not make use of a coordinator, each participant needs to have enough state information about the state of the overall computation to enable it to make the choices that would have been made by the coordinator on behalf of the participant. We use state alignment for this purpose: associated with the message sending operation there is a little bit of handshake synchronization involved that ensures that both the parties have a common understanding of the status of the operation termination (whether the message has been delivered and accepted by the receiver for processing or rejected). This is the basic coordination mechanism we need for composing executable atomic choreographies. We briefly described the tool support we have developed for automatically establishing conformance between a contact and a choreography and pointed out directions for further research.

REFERENCES

[1] OASIS, "http://www.oasis-open.org/committees/ws-tx/."

[2] P. Helland, "Life beyond distributed transactions: an apostate's opinion," in *Proc. 3rd Biennial Conf. on Innovative Data Systems Research (CIDR'07)*, 2007, pp. 132–141.

[3] C. Bussler, "P2p in b2bi," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE, January 2002.

[4] U. M. Visweswara, A. Gohad, and P. S. Rao, "Out-of-the-enterprise b2b gateway cloud service for emerging markets," in *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, October 2012, pp. 1–6.

[5] P. Greenfield, D. Kuo, S. Nepal, and A. Fekete, "Consistency for web services applications," in *Proceedings of International Conference on Very Large Data Bases (VLDB'05)*, Tromso, Norway, August 2005, pp. 1199–1203.

[6] T. Wang, J. Vonk, B. Kratz, and P. Grefen, "A survey on the history of transaction management: from flat to grid transactions," *Distribted and Parallel Databases*, vol. 23, pp. 235–270, 2008.

[7] I. Houston, M. C. Little, I. Robinson, S. K. Shrivastava, and S. M. Wheater, "The corba activity service framework for supporting extended transactions," *Software: Practice and Experience*, vol. 33, 2003.

[8] M. P. Papazoglou and B. Kratz, "Web services technology in support of business transactions," *Service Oriented Computing and Applications*, vol. 1, pp. 51–63, 2007.

[9] C. Sun, E. el Khoury, and M. Aiello, "Transaction management in service-oriented systems: Requirements and a proposal," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 167 – 180, 2011.

[10] S. Finkelstein, T. Heinzel, R. Brendle, I. Nassi, and H. Roggenkemper, "Transactional intent," in *5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, Asilomar, CA, January 2011, pp. 104 – 113.

[11] J. D. Tygar, "Atomicity in electronic commerce," in *15th Annual ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, 1996, pp. 8–26.

[12] C. Molina-Jimenez, S. Shrivastava, and M. Strano, "A model for checking contractual compliance of business interactions," *IEEE Trans. on Service Computing*, vol. PP, no. 99, 2011.

[13] C. Molina-Jimenez and S. Shrivastava, "Establishing conformance between contracts and choreographies," in *15th IEEE Conference on Business Informatics (CBI 2013)*.

[14] A. Berry and Z. Milosevic, "Extending choreography with contract constraints," *International Journal of Cooperative Information Systems*, vol. 14, no. 2, pp. 131–179, 2005.

[15] G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," in *10th Int'l Enterprise Distrib. Object Computing Conf. (EDOC'06)*. IEEE CS, 2006, pp. 221–232.

[16] A. K. Chopra and M. P. Singh, "Multiagent commitment alignment," in *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*.

[17] P. R. Telang and M. P. Singh, "Specifying and verifying cross-organizational business models: An agent-oriented approach," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 305–318, 2012.

[18] ebXML, "ebxml business process specification schema technical specification v2.0.4," 2006. [Online]. Available: http://docs.oasis-open.org/ebxml-bp/2.0.4/OS/spec/ebxmlbp-v2.0.4-Spec-os-en.pdf

[19] RosettaNet. [Online]. Available: http://www.rosettanet.org/

[20] L. Wang, A. Wombacher, L. F. Pires, M. J. van Sinderen, and C. Chi, "A state synchronization mechanism for orchestrated processes," in *16th International Enterprise Distributed Object Computing Conference (EDOC 2012)*, Beijing, China, September 2012, pp. 51 – 60.

[21] M. Autili, D. D. Ruscio, A. D. Salle, P. Inverardi, and M. Tivoli, "A model-based synthesis process for choreography realizability enforcement," in *16th International Conference on Fundamental Approaches to Software Engineering (FASE)*, ser. Lecture Notes in Computer Science, vol. 7793, 2013, pp. 37–52.

[22] SAVARA, "Savara and testable architecture," http://savara.jboss.org.

[23] RosettaNet, "Rosettanet methodology for creating choreographies," July 2012, version Identifier: R11.00.00A. [Online]. Available: http://www.rosettanet.org/

[24] A. Schönberger, "Visualizing b2bi choreographies," in *Proc. IEEE Int'l Conf. on Service-Oriented Computing and Applications (SOCA'11)*, 2011, pp. 1–8.

[25] OTA, "Open travel alliance." [Online]. Available: http://www.opentravel.org/

[26] C. Molina-Jimenez and S. Shrivastava, "Maintaining Consistency between Loosely Coupled Services in the Presence of Timing Constraints and Validation Errors," in *Proc. 4th IEEE European Conf. on Web Services (ECOWS'06)*. IEEE CS, 2006, pp. 148–160.

[27] C. Molina-Jimenez, S. Shrivastava, and N. Cook, "Implementing business conversations with consistency guarantees using message-oriented middleware," in *Proc. 11th IEEE Int'l Enterprise Computing Conf. (EDOC'07)*. IEEE CS, 2007, pp. 51–62.

[28] A. Tanenbaum, *Computer Networks*. Prentice Hall PTR, 2003.

[29] OMG, "Bpmn v2.0 specification." [Online]. Available: www.bpmn.org

[30] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services*. Springer Berlin Heidelberg, 2004.

[31] A. Abdelsadiq, C. Molina-Jimenez, and S. Shrivastava, "A high–level model–checking tool for verifying service agreements," in *Proc. 6th IEEE Int'l Symposium on Service–Oriented System Engineering (SOSE'2011)*, 2011.

[32] C. Molina-Jimenez and W. Sun(Jim), "Deployment of mosco- a bpmn verifier: (users guide)," https://github.com/carlos-molina/mosco.git, 2012.

[33] G. J. Holzmann, *The Spin model checker: primer and reference manual*. Addison–Wesley Professional, 2003.

[34] G. Salaun, T. Bultan, and N. Roohi, "Realizability of choreographies using process algebra encodings," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 290–304, 2012.

[35] J. Eder and A. Tahamtan, "Temporal conformance of federated choreographies," in *DEXA 2008*, vol. LNCS 5181. Springer-Verlag, 2008, pp. 668–675.