Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions

Andreas Schönberger and Guido Wirtz
Otto-Friedrich-University Bamberg
Distributed and Mobile Systems Group
Feldkirchenstr. 21, 96052 Bamberg, Germany
{andreas.schoenberger | guido.wirtz}@wiai.uni-bamberg.de

Abstract—*Business process integration across enterprise boundaries is a complex task. Personnel from different enterprises lacks a common understanding of domain-specific terms or the essentials of a business process. Modeling support using not too complex descriptive formalisms is crucial for communication purposes when combined with suitable abstraction techniques for managing complexity. System validation requires a precise formal semantics for extending ad hoc analysis with formal methods and for directly translating and executing the provided models. We propose to model such collaborations as distributed services from a centralised as well as a distributed perspective to separate business logic from implementation. The centralised perspective is modeled with UML activity diagrams which are validated with model checking techniques. The distributed perspective is implemented using Webservice technology like WS-BPEL and WSDL but is also abstracted into a form suitable for model checking to evaluate whether the distributed perspective is a consistent and correct implementation of the global perspective.*

Keywords: B2B modeling, SOA, software validation, formal methods, distributed

1. Introduction

Business-2-Business (B2B) integration is a key success factor for enterprises nowadays. Ever growing competition forces reengineering business processes and integrating business processes among partners. Using Web Service technology, there are means for supporting these integration needs electronically. However, besides improving Web Service technology there's still a lot of research to do in the integration field. Modeling support as a key requirement arises from the characteristics of B2B integration. First of all integration projects are conducted by personnel from different enterprises. To address different background and technical skills of this personnel models are needed for communication purposes. Apart from that B2B integration frequently needs truly distributed computing because there's no central infrastructure installed for two particular enterprises or because business politics prohibit such central infrastructure. As distributed computing is a challenging task, special modeling support is needed.

From our point of view validation in early design phases (cf. [1]) is a key success factor in building complex systems. Design errors are costly if they are detected late in the development cycle. Model checking based on formal system models is a technique that supports detecting errors by completely exploring the state space of a system. Hence, errors that emerge from very improbable situations are more easily found. The modeling and verification approach presented here has been evaluated by means of an extensive case study on choreographing so-called RosettaNet *Partner Interface Processes* (PIPs) [2], [3]. RosettaNet is a non-profit standards organisation dedicated to supporting B2B integration and endorsed by over 500 companies worldwide. RosettaNet uses technology and ideas from Open-edi ([4]), UN/CEFACT Modeling Methodology (UMM, [5]) and ebXML [6]. An important part of the standard are the Partner Interface Processes (PIPs) that describe the application context, the content and the parameters for the electronic exchange of business documents. The PIPs are classified according to domain-specific criteria based on the purpose the interaction specified has in the overall B2B process. RosettaNet suits well as a test case because it is a standard itself and a major part of its standard is devoted to message contents of business collaborations, an important task that is complementary to the aspects addressed here.

This paper introduces the basics of our approach, sketches the modeling techniques used with an emphasis on the global view, defines a classification for interesting properties to be checked and gives an example from modeling a simple B2B collaboration in a standardised way. The example used to illustrate some of our ideas in this paper is an excerpt of the case study [7]. It describes some of the steps needed to negotiate a contract between two prospective business partners. The entire interaction is composed of 9 different PIPs that stem from the segment *3A Quote and Order Entry* of the cluster *3 Order Management* according to the RosettaNet classification system. The overall goal of the composition is the negotiation of a contract and of contract changes. A more concise overview of the case study can be found in [8].

published in: Proc. of SERP'06, Las Vegas, June 26th-29th 2006, CSREA Press 2006

2. The Core of the Approach

Because B2B interactions consist of different, often changing, partners interacting in perhaps complex ways, there are conflictive needs, each modeling approach for this area has to address. To organise the entire collaboration a common protocol for all participating roles is essential in order to define for all partners how interaction is assumed to happen, i.e. who is allowed to initiate a conversation, who is obliged to answer within a given time and so on. On the other hand, the details how a single partner implements its part and how this is embedded in the local organization should be kept unknown to the outside. So, each partner can keep his business secrets. Hence, a clear distinction between these views is an advantage for a B2B modeling approach. The same holds also for the tools and description techniques used. Whereas the global perspective should be understandable to all partners, it should use a really simple modeling language whereas the local perspective is meant for designers and implementers of distributed complex systems and hence, can be much more technical. However, there has to be a link between the two views in order to ensure that the different models can be combined to an overall, locally als well as globally, working model. Therefore we propose the following approach to building up business collaborations.

1. Modeling a business collaboration from a centralised perspective. Communication between personnel from different enterprises can be supported by first focussing on the business logic of the collaboration. The so-called *centralised perspective* (CP) specifies the abstract business state of the collaboration, the events that trigger state changes, so-called micro-choreographies that consistently perform state changes and the control flow of the collaboration. Interpreting the CP as the common view of all collaboration participants on the collaboration progress is key to understanding the CP. Thus the state under consideration reduces to exactly those facts the collaboration participants have to agree upon and micro-choreographies can be interpreted as single actions that change state leaving out message passing details. This modeling metaphor makes modeling quite simple and does not require technical experts, who possibly don't know business logic very well, to create the model. Nevertheless the modeling technique applied should have clear semantics to avoid misunderstandings between collaboration participants and to provide the foundation of (semi-) automated generation of a distributed implementation. The model of the CP then gives context to a distributed implementation of the collaboration which is a similar approach as pursued by WS-CAF/WS-Context ([9]).

2. Modeling a business collaboration from a distributed perspective. The *distributed perspective* (DP) models the implementation of the CP in a distributed environment, i.e., represents abstract business states and specifies protocols for performing micro-choreographies that ensure distributed consensus. Complexity is handled in two different ways. The global view on business logic is already fixed when modeling the DP and the concept of micro-choreography helps in unitising the implementation model. Further tasks to be fulfilled on the DP are achieving agreement on the start of micro-choreographies, proving that the DP conforms to the CP and integrating local business politics of the collaboration participants. The details of the local applications are encapsulated by means of a so-called internal process interface. This adapter is the interface for detecting events of the real world and changing the real world as well as generating and interpreting business documents.

3. Applying model checking techniques to ensure robustness. Business collaborations possibly exchange goods and services of considerable value. Therefore a robust design is needed in order to avoid losses. This means, the protocol to perform a collaboration is ideally defined in such a way that errors are only possible by violating the protocol thus identifying a responsible person to call to account for losses. This goal requires to look at all possible protocol runs for a given environment. As the set of possible runs in a concurrent environment grows quickly, time-consuming and error-prone manual analysis should be extended by automated analysis techniques such as model checking. Model checkers compute all possible runs of a given protocol and offer the possibility to check properties. Model checking is especially suited to perform validation in early design phases as models tend to be relatively small at that abstract stage and undetected errors cause high costs. Major problems in applying model checking techniques are the identification of relevant properties to check. Therefore we discuss a taxonomy of relevant properties of business collaborations. and identify requirements for model checkers as well.

In practice, such an approach must be supported by suitable technologies and tools. As the relationships between enterprises are numerous, i.e. enterprises have many partners, and dynamic, i.e. enterprises acquire and lose partners, these technologies and tools should be based on standards. Standards support quick and low-cost automation of business collaborations. We propose UML activity diagrams ([10]) and WSBPEL ([11]) as enabling, but not exclusive, technologies for modeling the CP and the DP respectively. We found that UML activity diagrams [10] are a good choice for several reasons. UML activity diagrams offer a visual notation suitable for modeling business processes (cf. [12]) which supports the communication functionality of the CP. There is a large user community using activity diagrams so that a modeling approach based on activity diagrams can easily be adopted in practice. Even the RosettaNet standard used in our case study utilises a variant of UML 1.5 diagrams to describe the so-called *Business Operational View* of PIPs, i.e. who initiates an interaction etc. WSBPEL and in particular Web Services are platform and programming language independent which is impor-

tant for connecting heterogeneous systems. Moreover, WSBPEL seems to become the converging standard for describing Web Sercice choreographies.
Because model checkers themselves are not in the centre of our approach but rather are tools to be used and tested with respect to their usability, we investigated the tools freely available at the moment. For the interested reader more information about model checking can be found in [13]. A core requirement for practically applying model checking in our case is that the model checker in use can be applied more or less directly to the models of the CP and DP or, at least, to enhanced models of the CP and DP because having to model the collaboration twice would be too errorprone. We decided to use TCM/TATD[1] for validating the CP and SPIN[2] for validating parts of the DP. The main reason for the former was the ability to transform (restricted) UML activity diagrams automatically into a model to be checked by the tool. SPIN with its input language Promela has been used because the WS-BPEL process model is not too far from the notions of Promela [14].

3. Modeling the Centralised Perspective

One important factor of our modeling approach is the insight that state is crucial for modeling business collaborations. The goal of any business collaboration is achieving a new and common state, e.g. signing a new contract. Moreover the applicability of transactions and the way transactions are conducted within a collaboration may depend on what steps have been taken before. The exchange of an order of goods may require a quote to be exchanged before and refer to prices that are fixed in the quote. We *model state explicitly in our approach and define it as the common view of the collaboration partners on the progress of the collaboration* (process state). The progress itself can be interpreted as an abstract business state, e.g. the state of having exchanged a quote. A collaboration can then be modeled as a set of process states, that are controlled by control flow constructs. Clearly, a common view on progress is hard to implement in a truly distributed environment but considering the use of distributed commit protocols it can be achieved that each participant is at most one step behind the other. However, we do not introduce states in which the participants have a different view on progress, but make the obligation that process states can only be modified by distributed communication primitives that ensure a consistent change of the local views of the participants. This is what we call *micro-choreographies* because the collaboration participants have to exchange a set of messages according to some strict rules for implementing consistent changes of process states. Having process states and micro-choreographies defined we still need means to manage the control flow of collaborations, i.e. constructs like decision, loop, parallel flow or synchronization and we still need events to capture changes in the real world that trigger the execution of micro-choreographies.

There are some assumptions about the use of modeling elements which require a specific use of RosettaNet specifications. Most importantly, the PIP execution protocol has to be restricted so it guarantees distributed consensus. When using PIPs according to the RosettaNet standard there are rare cases that lead to diverging views of the participants. That contradicts our assumption that micro-choreographies must change process states in a consistent way. By executing a two-phase-commit protocol (e.g. [15]) at the end of each PIP, consensus whether all business documents have been exchanged successfully or not is ensured. An assumption essential to all modeling approaches is the restriction to finitely representable models. In our case, the use of RosettaNet business documents has to be restricted in a manner that the evaluation of business documents can be represented by a finite set of values, because these values are used to route the control flow of interactions. As some RosettaNet documents may contain arbitrary large data structures like lists of purchase order items this assumption is not met a priori. One possibility to solve this problem is to apply a function to business documents that computes a finite set of values for routing purposes. To keep models handy this set should be quite small which in turn may enforce some restrictions on the use of business documents. A simplifying restriction we used in our case study is that either all items of a purchase order may be accepted or none at all. Our approach for the global perspective can easily be mapped to activity diagram elements as follows.

- *Process states are represented by state machine states* as activity diagrams are a special case of state machines in UML 1.5. Such states may be hierarchically composed. Thus all attributes that make up the process state of collaborations are conveniently modeled as substates of the state machine state. These may have multiple incoming and outgoing transitions. Figure 1 shows some process states of our case study with relevant attributes.
- *Micro-choreographies are modelled as activities where a single activity models a whole micro-choreography*. In our case study, activities are used to represent whole PIP executions, i.e. PIPs are interpreted as micro-choreographies that consistently change process states. PIPs are related to activities by using their names as part of the activity name. The activity name is further extended by the role names of the collaboration participants, where the name of the role that initiates the PIP goes first. Micro-choreographies always have exactly one incoming and one outgoing transition. Figure 1 shows PIP *3A1 Request Quote*.

[1] http://wwwhome.cs.utwente.nl/~tcm/tatd.html
[2] http://spinroot.com/spin/whatispin.html

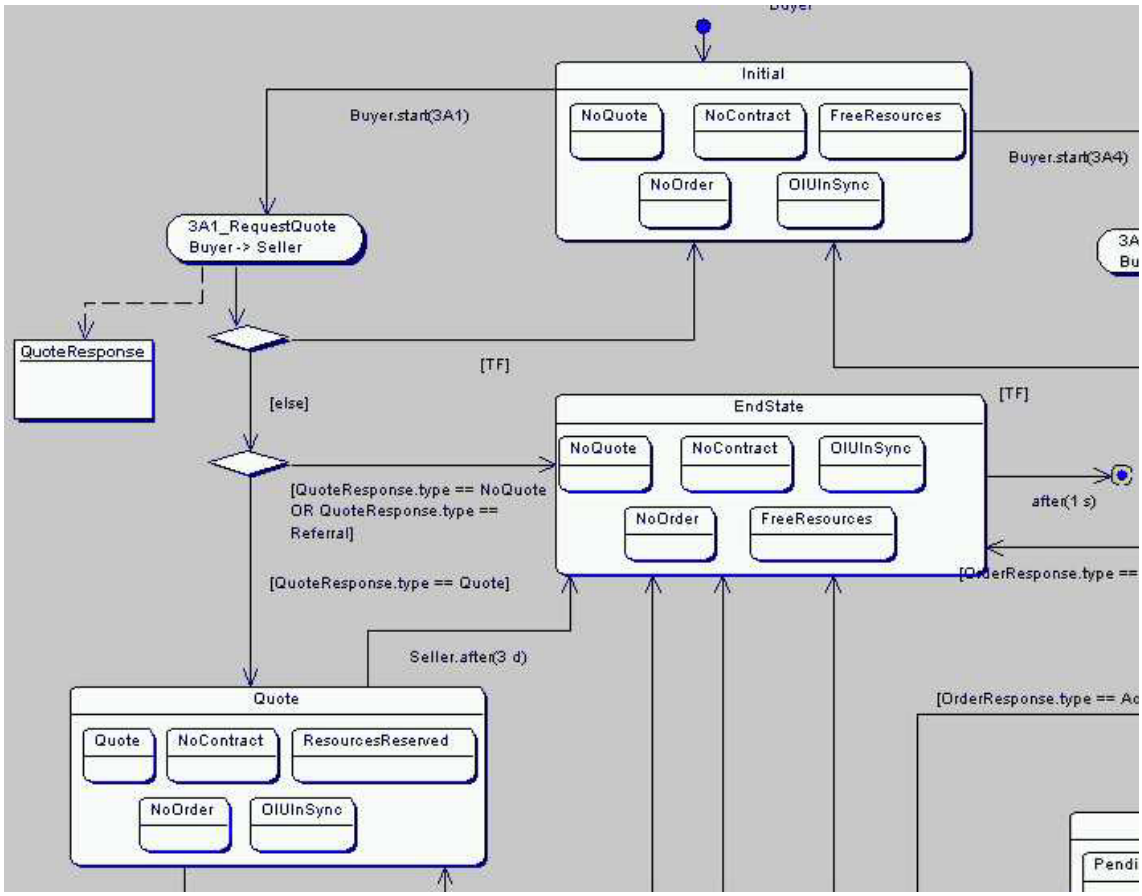

Fig. 1. Initial steps: negotiating a contract

Control flow is supported by activity diagrams with various node types. Alternative flows can be visualised by decision and merge nodes whereas parallel flows can be visualised by fork and join nodes. *Pseudo states* are used as usual in activity diagram modeling, i.e. start and end nodes as well as decision and merge nodes.

- *Transitions describe control flow by connecting process states and micro-choreographies*. They also ship with events, guards and actions. Events capture changes in the real world, esp. the triggering of a micro-choreography. In principle events are always local to one collaboration participant as events for both participants would require distributed consensus themselves. Exceptions to this principle can only be made if distributed consensus has been achieved beforehand, e.g. it can be agreed upon the reservation time of a resource while agreeing upon the reservation itself (refer to [7] for more details). Guards can be used to route between alternative flows of execution. *To achieve a common view on collaboration progress all participants of a collaboration have to take the same routes which means that the distributed environment has to be considered for guards as well, i.e. guards may only refer to variable values that have been agreed upon by distributed consensus.*

Outgoing transitions of a process state always have an event that shows which real world event leads to the execution of the transition. They always end in a micro-choreography or in another process state. In the first case the name of the accompanying event starts with the name of the role that initiates the micro-choreography (e.g., `Buyer.start`). The name is then extended by an identifier for the local event of the role. In the second case the accompanying event represents a distributed timeout that is used to release resources of one of the roles. The name of that event starts with the name of the role whose resources are released and then specifies the time period that triggers the distributed time-out measured from the entrance time of the source process state (e.g., `Seller.after(3 d)`).

The outgoing transition of a micro-choreography ends in a decision process that determines the next process state depending on its outcome. Therefore the outcome is divided into a technical and a business result. If the execution of a PIP fails technically the process always returns to the process state from which the PIP was initiated. This case is represented as a guard with the constant *technical failure* (TF). If the execution of a PIP succeeds technically, the next process state depends on the (abstract value of the) content of the last business document that has been exchanged. This circumstance is represented by using the name of the last business document as the initial part of the variable names that are evaluated in guards to determine the

next process state. The restriction that there may only be one incoming transition to a micro-choreography is made to guarantee an unique process state to fall back to in case of a TF. Hence, the same PIP may be modeled multiple times within a collaboration.

- *Roles are represented by Swimlanes.* Roles are a means to specify the tasks a collaboration participant is obliged to fulfil. Swimlanes are used to visualise roles of collaboration participants because they are intended to organise responsibility for actions and subactivities ([10] section 3.89.1, p.3-161)). We put each activity a role potentially initiates into its swimlane. Each participant can take at most one role.
- *Business documents represented by object flows store the information that is negotiated during a collaboration*. Object flows reference the structural definition of such documents and thus abstract from their details. We used object flows to model the *last* business document that is exchanged during a micro-choreography because the content of that document is crucial for control flow routing. The use of object flows is a potential link to a structural view on the collaboration by modeling the structure of business documents as classes.

4. Checking a Collaboration Model

With our approach combining a centralised and a distributed perspective, different aspects of applying model checking become apparent. First of all, *sanity of the centralised model* should be rigorously checked because errors at this early step are most costly. In essence, sanity of the CP amounts to a sensible order of process states, PIPs as well as local events and distributed timeouts (refer to 4.2 for details). Second, *conformance of the distributed implementation with the centralised view* as well as proving it to be functional in its usage of WSBPEL and Web Services is to be ensured. Last, but not least, the *influence of local business politics of the collaboration participants* like, e.g., detecting events, that clearly affect properties of the CP like absence of deadlocks and livelocks, should be checked. Checking the internal process adapter w.r.t. proper abstraction against the local applications is still ongoing work. For the rest of this section, we focus on the most important aspect, the overall centralised perspective.

4.1. What can be checked ?

Before using model checking, one should be aware of its restrictions. Analysing a given property in a given model always depends on the amount of information present in that model. Hence, most of the time there are interesting properties that cannot be checked. In our case this may stem from the semantics of business documents that is not captured in detail in our model. To give an example, it would probably be illegitimate if a seller does not accept the purchase order of a buyer if the buyer exactly obeys the rules of a quote he has received from the seller beforehand. A less apparent case of properties that cannot be analysed with our model can be encountered when looking at events. Imagine a process state in which a buyer can initiate two different micro-choreographies, one to notify the success of a collaboration and one to cancel the whole collaboration. Assume further that the buyer has detected success and hence triggers a local event to notify his collaboration partner. Then he surely would not try to cancel the collaboration afterwards. In our model there's no information if such event sequences are admissible or not. These problems induce that the systems we model allow for more executions than possible in the real world. This has consequences depending on what kind of property is analysed, i.e. if the property at hand is a *safety* or a *liveness property*. The truth value of a safety property can be decided in any particular state of the state space only by looking at the sequence of states that led to that particular state. If this is not the case, i.e. if the truth value depends on future states, the property under consideration is a liveness property (cf. [13], p.84). Successfully verifying a safety property means that the property also holds in the real world whereas not being able to do so not necessarily means the property does not hold. Regarding liveness properties theoretically no results at all can be carried over to the real world from verifying or falsifying a given property in the model. Accepting these restrictions we still claim that model checking is useful for finding errors in our model.

4.2. What should be checked?

As the identification of properties to check, probably one of the most important parts in validating systems, is not inherently supported by model checking, validating systems still requires the validator to have a lot of experience and analytical skills. We propose a classification of properties to help the validator in finding the right properties to check. *Generic properties* refer to the process nature of a model and should hold in any model. Frequently demanded properties are the absence of livelocks and deadlocks as well as that processes should always terminate in a defined end state. Moreover, any state of a process model specified by an activity diagram should be reachable and any transition of the model should be fired in at least one run (cf. [16]). Finally, in activity diagrams *lack of synchronization* should be impossible, i.e. there should be no situation in which two instances of the same node are active [17]. *Domain specific properties* refer to the details of B2B integration. Most business processes require resources to be reserved. So one important property is that at the end of any business process, all resources should be free. A further property is that a state should only be reachable if all its substates have been created by an appropriate micro-choreography execution or by a distributed timeout, i.e. there should be no state changes without distributed consensus.

From our point of view the creation of a more comprehensive set of properties is the task of B2B standards consortia such as RosettaNet. [18] defines property patterns that could be applied to find the right properties. These patterns are *Sequence*, *Combined Occurrence/Exclusion*, *Consequence* and *Precedence*. *Application specific properties* cannot be defined before a specific application context is given, but finding properties can be supported by looking at the purpose of a collaboration and by applying the patterns just named. One property that should hold in our use case is that a process state with a signed contract should only be reachable if the required *PIP 3A4 Request Purchase Order* has been executed successfully.

to models TATD offers the IN(*node name*) predicate to state that a node with name *node name* is part of the current configuration and the usage of boolean guard variables. Regarding our classification of relevant properties, TATD checks the reachability of every node and the possibility to fire every transition by default. Other generic properties like absence of deadlocks have to be encoded with temporal logic. For example, the formula $G\ (FINAL\ \vee\ (X\ true))$ encodes absence of deadlocks by stating that at any point in time (G) either an end state is reached (*FINAL*) or there is a transition to the next state (in the neXt state (X), the constant *true* holds). If a property does not hold, TATD computes and displays a counter example in red and blue colour.

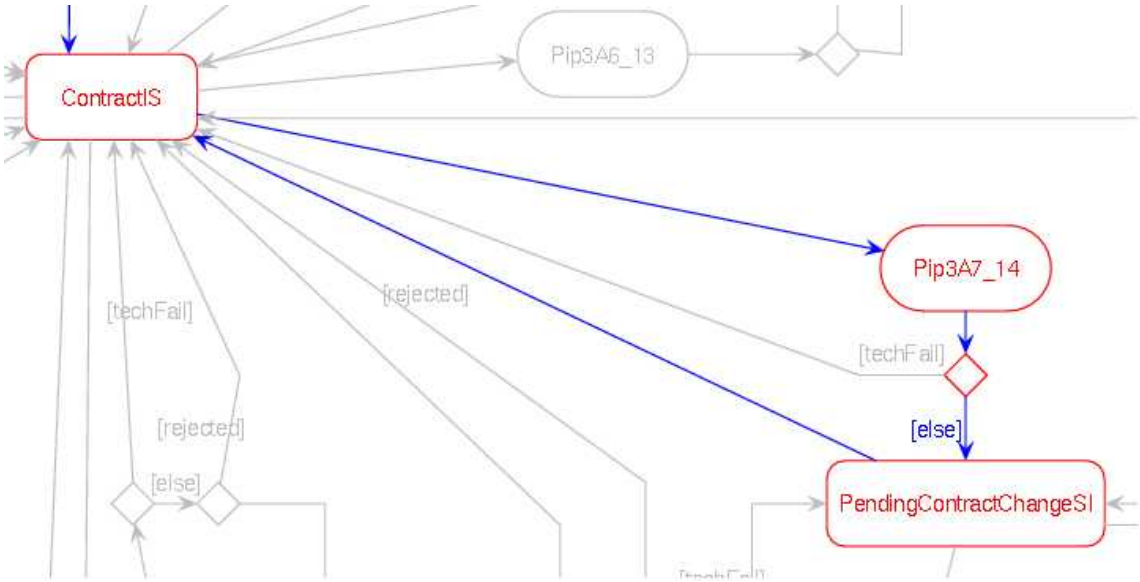

Fig. 2. A counter example showing a loop

4.3. Experiences with checking the global perspective

The sanity of the global perspective has been checked using TCM/TATD because of its ability to transform (restricted) UML activity diagrams automatically into a model to be checked by the tool [16]. As a prototype tool TATD does not fully meet some important requirements like simulation capabilities or resource usage feedback but it is definitely helpful for evaluation purposes. The semantics of activity diagrams offered is based on the notions of *configuration* and *step*. A configuration is a bag of activity and state nodes that are active at a certain point in time. Steps transform configurations in other configurations by computing and firing maximal sets of transitions that can be fired in parallel. The computation of new configurations is based on the presence of events and the valuation of guards. Non-determinism is used to simulate events of the outside world and to determine the value of guards. The state space is explored by computing all possible sequences of configurations. TATD offers two quite expressive kinds of temporal logic, namely subsets of PLTL and CTL (c.f. [13]) for formalising properties. To relate formulae

Figure 2 shows a possible loop that is highlighted. The validation of domain specific and application specific properties can be done with TATD in theory but in practice the formalisation of properties is clumsy because the query language of TATD is not adapted to our specific needs, i.e. states and PIPs cannot be typed and there's no predicate for referring to a PIP execution with a certain outcome. We refer to [7] for detailed examples describing domain-specific as well as application-specific properties.

5. Related work

First of all we want to emphasise that the idea of transforming consistent states into consistent states by means of transactional activities is well-known from database theory. But we apply this concept in the domain of B2B integration by defining abstract business states as the common view of collaboration partners on the progress of the collaboration. In the centralised perspective we interpret the whole collaboration as one single process. An approach that is quite similar to ours in the sense of defining a context for distributed implementations can

be found in ([9]), but the definition of context is left unspecified. A lot of related work is done by the Web Services community with respect to composing services (e.g. [19], [20]). The composition of services definitely is necessary for integrating businesses but most approaches act on the level of single service calls and not on the level of transactional micro-choreographies as we do.

Most of the concepts used for the centralised perspective can be *emulated* by the concepts introduced in the recent WS-CDL 1.0 [21] recommendation but the main difference is the level on which the approaches operate. The centralised perspective of our approach requires the modeller to think in micro-choreographies, process states, events and distributed time-outs whereas WS-CDL requires the modeller to think in WSDL interfaces, message exchanges and many more technical details. So it is reasonable to think of the centralised perspective more in terms of a business process and to think of WS-CDL more in terms of technical specification.

The new ebXML BPSS v2.0.1 standard[3] has the biggest similarities to what we do. The BPSS models a collaboration based on *Business Transactions* that exchange business documents and then composes *Business Collaborations* out of Business Transactions. Business Collaborations can be composed hierarchically. BPSS Business Transactions are quite similar to our micro-choreographies and BPSS Business Collaborations resemble our compositions of micro-choreographies. However, BPSS Collaborations do not explicitly model abstract process states as the common view of the collaboration participants on the progress of the collaboration. Moreover BPSS does not define a mapping to UML activity diagrams or any other visual modeling language.

Looking at RosettaNet as our use case, [22] have proposed a framework for executing multiple PIPs, but they did not define a modeling approach for creating PIP compositions. Hofreiter and Huemer [23] further proposed the translation of PIPs to BPEL but they do not consider the influence of state on business collaborations the way we do and they didn't use a model checker for validation purposes. Finally there are some taxonomies of properties for model checking properties ([18], [16]), but these differ from ours in being not that detailed or are defined for a different application area.

6. Conclusions and Future Work

We have outlined a two-level modeling approach to model the control flow of B2B interactions in a manner that is suitable for implementation in the context of orchestrating web services as well as for rigorous formal reasoning through the application of model checking tools. The approach has been evaluated by means of a real-life case study.

Future work is focussed on making the modeling approach more accurate by reducing some of its current restrictions, esp. by incorporating resource considerations for executing tasks as well as the data inside business documents which incorporates the structural view. Moreover, the integration of model checking with visual modeling in a manner useful for non-experts requires understandable annotations for desired properties and results of the analysis. Visualizing errors within the model is an important step here. Last, but not least, support for automatically generating implementations is still limited, but an important acceptance factor.

[3] `www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-bp` is not yet an official specification

References

[1] Holger Giese and Guido Wirtz. Early Evaluation of Design Options for Distributed Systems. In *Int. Symp. on SWE for Parallel and Distributed Systems (PDSE'2000), Limerick, Ireland*. IEEE Press, June 2000.

[2] RosettaNet, www.rosettanet.org. *RosettaNet Implementation Framework: Core Specification*, v02.00.01 edition, March 2002.

[3] Suresh Damodaran. B2B integration over the Internet with XML: RosettaNet successes and challenges. In *Proc. of the 13th Int. WWW Conference*, pages 188–195, New York, 2004. ACM Press.

[4] ISO/IEC. *Information technology - Open-edi reference model*. ISO/IEC, 2 edition, May 2004.

[5] UN/Cefact. UN/CEFACT' s Modelling Methodology N090 Revision 10, November 2001.

[6] ebXML. http://www.ebxml.org/.

[7] Andreas Schönberger. Modelling and Validating Business Collaborations: A Case Study on RosettaNet. TR 65, Bamberg University, 2006.

[8] Andreas Schönberger and Guido 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*, 2006.

[9] OASIS Open. Web Services Composite Application Framework (WS-CAF), 2003.

[10] OMG. *OMG Unified Modeling Language Specification*. Object Management Group, Inc., 250 First Ave. Suite 100 Needham, MA 02494, U.S.A., 1.5 edition, March 2003.

[11] IBM, BEA Sys., Microsoft, SAP AG, Siebel Sys. *Business Process Execution Language for Web Services 1.1*.

[12] M. Dumas and A. H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. *LNCS*, 2185, 2001.

[13] B. Berard, M. Bidoit, and A. Finkel et. al. *Systems and Software Verification : Model-Checking Techniques and Tools*. Springer-Verlag, Berlin, 1 edition, August 2001.

[14] Gerard J. Holzmann. *The SPIN Model Checker*. Addison-Wesley Pearson Education, September 2003.

[15] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, Upper Saddle River, NJ, 2002.

[16] H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, Univ. of Twente, Netherlands, 2002.

[17] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000.

[18] W. Janssen and R. Mateescu et. al. Model checking for managers. In *Proc. of the 5th/6th Int. SPIN WS*, pages 92–107. Springer, 1999.

[19] D. Beyer, A. Chakrabarti, and T. A. Henzinger. Web service interfaces. In *Proc. 14th Int. Conf. on WWW*, pages 148–159. ACM Press, 2005.

[20] David Skogan and Roy Gronmo et. al. Web Service Composition in UML. In *Proc. of the IEEE Enterprise Distributed Object Computing Conf. (EDOC'04)*, pages 47–57, 2004.

[21] W3C. *Web Services Choreography Description Language*. W3C, 1.0 edition, November 2005.

[22] A. Dogac et. al. An ebXML Infrastructure Implementation through UDDI Registries and RosettaNet PIPs. ACM SIGMOD Internat.l Conference on Management of Data, 2002.

[23] Birgit Hofreiter and Christian Huemer. Transforming UMM Business Collaboration Models to BPEL. In *Proc. of the OTM Workshop on Modeling Inter-Organizational Systems*, 2004.