Formal Based Correctness Check for ePASS-IoS 1.1 Process Models with Integrated User Support for Error Correcting

Stephan Borgert and Max Mühlhäuser

Technische Universität Darmstadt
Dept. of Computer Science
64289 Darmstadt, Germany
{stephan,max}@tk.informatik.tu-darmstadt.de

Abstract. To ensure the correctness of business process models, automatic and manual methods are applied. Since the manual checks are time consuming and expensive, the automatic methods should be as effective as possible. An established verification check is the check for the interaction soundness, i.e. the process model can be executed without deadlocks. Normally, these approaches compile the graph based models to expressions of a formal language which is passed on to a model checking tool for verification. The drawback with this methods is that the results are hard to use for analyzing the causes of errors. In this paper, we present an integrated approach that is able to find important error patterns, and supports the user in correcting errors while still having a high performance.

Keywords: Formal Verification, Interaction Soundness , Subject Oriented Modeling, PASS, π-Calculus.

1 Introduction

A model designer of the Metasonic[19] suite can move from a business process to a complete application in four steps: 1) Model 2) Evaluate 3) Integrate 4) Execute. The correctness of the models is checked in the steps 1) and 2) and in accordance to [[10], p.312, Sect. 16.3] two aspects of correctness are distinguished

- A system must have certain properties, e.g. livelock free, deadlock free which are independent of the application. This is implicit correctness. A deadlock free system is also called an *interaction sound* system in the context of service compositions. A formal definition is given in [25].
- A specified system must do what a designer has intended. This is explicit correctness.

Explicit correctness checks what does the process and implicit correctness checks how the process does it. To be exact: Does the process run without errors. Therefore, explicit correctness has a higher priority than implicit correctness

A. Nanopoulos and W. Schmidt (Eds.): S-BPM ONE 2014, LNBIP 170, pp. 20–40, 2014.

but explicit correctness is only given when implicit correctness is fulfilled. If a process is running into a deadlock, this should not fit the designers intents. Manual interrupts are no solution for this problem. Of course, the deadlock would be solved but if the model designer knows about a deadlock, the solving of the deadlock should be part of the process model. The same applies for timeouts. When they are only used to solve a deadlock but not to terminate the process model in a proper way, explicit correctness is not given. Explicit correctness is checked in step 2) by manual simulation, business users execute the model on an abstract level and simulate the behavior of the subject providers. Checking for explicit correctness has to be done manually since current systems are not able to interpret the semantics of the models. The effort for this should be as low as possible for three reasons. 1) Manual simulation is slow, 2) employees are expensive and 3) they still can overlook errors which could become expensive if they occur during runtime. To minimize the effort computer supported tools are used. The Metasonic Suite includes a check for structural soundness of process models. It checks for properties like: Is there exactly one start node per subject, is there at least one end node per subject, is there a path from every node to an end node and so on. This is very helpful to avoid many errors but there are still more error types the tool can not find. Since subject and service oriented business process models are also models of distributed systems, the verification for interaction soundness is a common correctness check.

The "Parallel Activity Specification Scheme" (PASS[9,12]) is the modeling concept implemented in the Metasonic Suite. An extended version of PASS was introduced in [11]. We denote this version as extended PASS or ePASS and a variant of it, focusing on support the internet of service paradigms, as ePASS-IoS[5].

Distributed systems suffer from 3 inherent issues. Firstly, non deterministic behavior can occur: every time a distributed system is executed, the behavior of the system can differ. The behavior is the interaction behavior i.e. the communication protocol among the involved sub systems. The reasons for this are the following: Not all subsystems of a distributed system work with the same speed at all run times and further more the time needed for conveying the messages can differ. Secondly, accessing the global state of the system is impossible. Monolithic systems are easier to analyze because e.g. debug mechanisms like break points can be used to detect errors. The same mechanisms are not suitable for distributed systems because even if it were possible to place break points in every sub system, they would not meet at the same time per system run because of the nondeterministic behavior. Therefore it would not be possible to stop the system in every run at the exactly same global state. Finally, the system time will be different on every subsystem: the system times will not be set to the exact same time. This is an important point for time critical systems, for business processes it is of less significance. Hence, it does not have to be considered for our purposes. All in all, error detection in distributed systems can be very hard and therefore the methods like verifying interaction soundness are supposed to be useful for ePASS process models as well.

The authors in [7] distinguish interconnection modeling and interaction modeling. Modeling the internal behavior of the participants and relating them with message flows is defined as interconnection modeling. Modeling the interaction behavior between the different participants is called interaction modeling. Therefore ePASS falls in the category interconnection models. BPMN supports interconnection and interaction modeling techniques. Both techniques support the modeling of choreographies but the expressiveness is different[6]. An example for interaction modeling can be found in [7].

Anti-patters were derived and used in [3,4,7]. In [3] anti-patterns are defined as "Anti-patterns describe undesirable constructs that may introduce errors or inefficiencies." The authors in [4] have stated: "Each anti pattern declaratively describes a violation scenario." In [7] a set of 8 choreography modeling anti-patterns were identified. These patterns were collected by observing people modeling BPMN and "can be observed in a large number of interconnection models." The anti-patterns can be expressed by interconnection models.

The paper is structured as follows: Firstly, we briefly introduce the ePASS-IoS language elements. Next, we motivate our decision to use π-calculus as formal foundation and introduce how we exploit it for the purpose of interaction soundness verification. After giving implementation details of the demonstrator, we evaluate our approach by applying it on the 8 different choreography anti-patterns. Then, the performance of the approach is discussed and finally, the summary and conclusion are given.

2 Verification of ePASS IoS Models

2.1 ePASS-IoS

Graphical Syntax. The graphical elements of the extended Parallel Activities Specification Scheme for the Internet of Services (ePASS IoS) 1.1 have been introduced in [5]. The graphical language set has been explained by graphical examples. We introduce a simplified notation in figure (1), in order to save space depicting the examples.

Subjects are an abstraction of actors. The execution of the subjects is done by concrete actors that are humans, machines or software services. These concrete actors are called subject providers. Rules have to be defined for every process to link the subject providers to the subjects during runtime. This late binding mechanism also has the advantage of decoupling the process models from the enterprise. More information can be found in [11,5]. Subjects are connected by unidirectional channels, which enables communication among subject providers by exchanging messages.

As shown in figure (1), we draw the internal behavior directly into the subjects. This is in contrast to other ePASS systems where two layers are used.

Send, **Receive** and **Action** are the basic activities. Send and Receive states are denoted with `(S)` and `(R)`, while the name of the action is written directly into the state symbol. The transition labels are *exit conditions*. *Action* states have the results of the action as *exit conditions*. When Action states only have

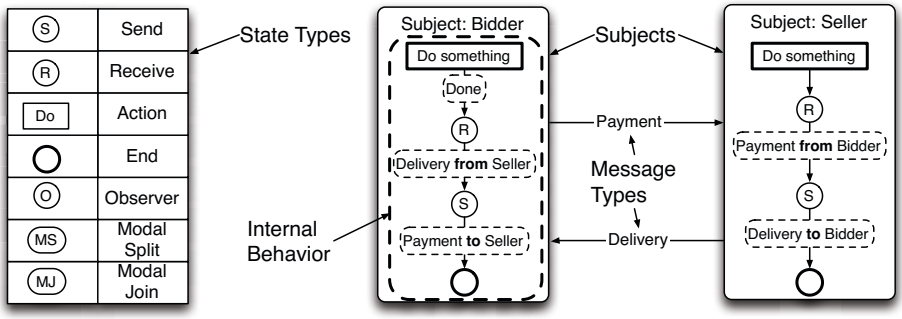

Fig. 1. Simplified notation of the ePASS-IoS 1.1 language elements

one exit condition, the exit condition label can be omitted. The exit conditions of **Send** states are (<Message-Type>, <Receiver>), written as <*Message-Type*> ***to*** <*Receiver*>. The exit conditions of **Receive** states are (<Message-Type>, <Sender>), written as <*Message-Type*> ***from*** <*Sender*>. The internal behavior of subjects have exactly one **Start** state and can have an arbitrary number of **End** states. Start states are denoted by a bold border. Every state can be a start state. The **Observer** manages interrupts and exceptions. In case of such an event, the control flow is lead to an alternative behavior. Start states can have an arbitrary number of out-transitions and the End state can have an arbitrary number of in-transitions. All other states can have an arbitrary number of in- and out transitions. The semantics of this pattern is an exclusive choice split and join. In order to fulfill a task, it is often the case that certain activities have to perform and others could optionally be performed in addition . This is why `(MS)` and `(MJ)`, which are called **Modal split** and **Modal join**, are introduced. They form a combination of an AND- split and join pair and an OR- split and join pair. These symbols are specified e.g. in the BPMN 2.0 [22] specification. Since the concerning semantics is an interleaving semantics, the parallel execution does not have to be performed in parallel. Thus, the human actors are still able to perform these kinds of behavior. In addition to the normal control flow transitions, **timeout transitions** can be used in PASS / ePASS. We would depict timeout transitions with a dashed line. Since the anti-patterns do not take timeouts into consideration, we are not using them in this paper. How ePASS timeouts can be handled concerning the formal verification can be found in previous work [5].

2.2 π Calculus as Formal Foundation

A formal semantics for the language elements is needed for the purpose of formal verification. A common way to do this is by encoding the elements with a suitable formalism. Many approaches of the BPM community use petri nets and

its variants for it. Others use process algebras, abstract state machines, or other state and transition based diagrams.

The used formalism should fit the verification requirements as well as possible. For the purpose of verifying ePASS IoS models for interaction soundness, the following requirements have been stated by us:

- As mentioned above, the purpose of the verification method is to perform a check for interaction soundness. It has to detect deadlocks.
- Further more it has to be internal and not external. An external one delegates the verification task to model checker tools which verify the models and return the results. The advantage of this approach is that it is easier to get a result. The disadvantage is, the results are not easy to interpret. The model checkers input consist of roughly two parameters. The first one is the model which has to be checked and the second one is the specification the model has to meet. The specifications are usually formulated in a formula from mathematical logic and therefore allows general purpose model checking. The results are difficult to interpret and it is not easy to analyze the reason of error sources automatically. The results have to be analyzed and new specifications have to be formulated; they have to be given to the model checker again and again until the error sources are found. A user friendly support for business users is hard to implement and therefore an internal check is required. In consequence, verification algorithms must be implemented into the modeling system. This way a user friendly analysis can be achieved.
- The correctness check must not take to long. We required that the check has to be faster then 15 seconds on an average computer.

Different diagram types can serve to model distributed systems like automata, state charts , FSM etc. One, very simple type, is the Labeled Transitions System (LTS).

Definition 1 (Labeled Transition System [14]). *Let* Act *be a fixed set of actions. A labeled transition system* $LTS = (\text{Proc}, \rightarrow)$ *over* Act *consists of*

- *A set* Proc *of states and*
- *A set* $\rightarrow \subseteq \text{Proc} \times \text{Act} \times \text{Proc}$ *of transitions between states.*

To model distributed systems, three different types of actions were introduced: send, receive and the internal action τ. These three actions are the base language elements of ePASS but the internal action τ is specified more precisely in the internal behavior. They can be converted to τ actions in order to anonymize the internal behavior. Thereby, an external representation is obtained which can still be used for verification purposes like deadlock, livelock checks or checks for equivalent interaction behavior of different subjects. Modeling distributed

Table 1. Syntax of the π-calculus. $\{x, y, z\}$ are *names* and τ the hidden action. The term $\overline{\mathrm{x}}\langle\tilde{y}\rangle$ denotes sendig $y_1, \ldots, y_n$ messages via channel x while $\mathrm{x}(\tilde{y})$ denotes receiving $y_1, \ldots, y_n$ messages via channel x.

Processes	P, Q	$::=$	
		$\mathbf{0}$	Inactive Process
		$\mid \pi.P$	Prefix
		$\mid P + Q$	Sum
		$\mid P \mid Q$	Parallel
		$\mid (z)P$	Restriction
		$\mid K$	Identifier
Prefixes	π	$::=$	
		$\overline{\mathrm{x}}\langle\tilde{y}\rangle$	Send
		$\mid \mathrm{x}(\tilde{y})$	Receive
		$\mid \tau$	Hidden Action

systems only with LTS is a very challenging task because the number of states of a distributed system grow very fast with the number of subsystems and the number of states of each subsystem. In this case a good choice to model the system is a process algebra. Process algebras consist of a set of actions and a set of operations defined on these actions. One of the simplest process algebras is the Calculus of Communicating Systems (CCS)[21]. Its actions are also send, receive, and τ. A more sophisticated and the most popular process algebra is the π-calculus[20,26]. It has as additional feature to CCS like a channel construct and a concept of mobility of channels, which allows dynamic restructuring of process models. The syntax of the channels is defined in table (1) and the semantics in table (2). The formalism should be as simple as possible to avoid needless complexity. Since CCS is simpler it should be the first choice. On the other side, two good reasons led to the choice for the π-calculus as formal foundation: 1) The π-calculus is more expressive than CCS. Hence, complex processes can often be modeled with less effort in π. 2) The use of π-calculus is necessary to support multi subjects. Although multi subjects are not supported in the current work, using π-calculus simplify extension offers. A term of process algebra actions and operations is called a process. The operations and the syntax are given in table (1) and the formal semantics is given in table (2). There are also three different types of actions: 1) send, receive and hidden actions. The send and receive actions are used for exchanging messages between processes. The hidden action is always donated by a τ and serves as abstraction of concrete actions. In this section a brief outline of the π-calculus is given since it is the foundation for this formal verification approach. The syntax of the algebra is given in table (1).

Inactive Process. The inactive process **0** does not do anything. It is a termination of a process.

Prefixes. The different prefixes define the different actions. A send action $\overline{\mathrm{x}}\langle\tilde{y}\rangle$ sends the value y via the channel x. Angel brackets stand for free values, namely y. Receive actions like $\mathrm{x}(\tilde{y})$ receive a value via the channel x and save the result in the variable y. A send action of one process can be synchronized with a receive action of another one by using the same channel. The result is a τ action which is also called hidden transition. It is an abstraction of what the transition really does.

Sum, Parallel, Restriction. The sum is an exclusive choice between the processes P and Q. $P \mid Q$ denotes the parallel execution of P and Q and $(z)P$ restricts the scope of the name z to P.

Identifier. The identifier enables the definition of names for processes.

Table 2. The reduction semantics of the π-calculus

$$\text{R-Inter}\ \frac{}{\left(\overline{x}y.P_1 + M_1\right)\left|\left(x(z).P_2 + M_2\right)\right. \rightarrow P_1|P_2\{^z/_y\}}$$

$$\text{R-Tau}\ \frac{}{\tau.P + M \rightarrow P} \qquad \text{R-Par}\ \frac{P_1 \rightarrow P_1'}{P_1|P_2 \rightarrow P_1'|P_2}$$

$$\text{R-Res}\ \frac{P \rightarrow P'}{(z)P \rightarrow (z)P'} \qquad \text{R-Struct}\ \frac{P_1 \equiv P_2 \rightarrow P_2' \equiv P_1'}{P_1 \rightarrow P_1'}$$

$$\text{R-Ident}\ \frac{P \rightarrow P'}{K \rightarrow P'} K = P$$

The semantics is defined by structural operational semantics rules (SOS) [23] of the general form $Rule = \frac{premises}{conclusion} condition$. The prefix "R-" denotes reduction. To give an example, the rule R-Par can be written as:

$$\textbf{if}(P_1|P_2\ \textbf{AND}\ P_1 \rightarrow P_1')\ \textbf{then}\quad P_1|P_2 \rightarrow P_1'|P_2$$

Let $P = \left(\overline{x}y.P_1 + M_1\right)$ and $Q = \left(x(z).P_2 + M_2\right)$. $\overline{x}y$ is a send prefix and $x(z)$ a receive prefix. Therefore P can send y via the channel x to Q where the variable z is assigned to y. In consequence all z terms of P_2 are renamed to y which is denoted by $P_2\{^z/_y\}$. The prefixes are consumed. The sub-processes M_1 and M_2 of P and Q are rendered void. In summary $P|Q$ evolves to $P_1|P_2\{^z/_y\}$.

A variant of the conclusion of R-Inter would be: R-Inter2-conc $\left(\overline{x}y.P_2 + M_2\right)\left|\left(x(z).P_1 + M_1\right)\right. \rightarrow P_2\{^z/_y\}|P_1$. We do not have to define a second rule for this variant because of the R-Struct rule: from $P_1 \equiv$ and $P_2 \rightarrow P_2'$ and $P_2' \equiv P_1'$ infer $P_1 \rightarrow P_1'$, where $\equiv$ is a structural-congruence relation. It is possible to infer R-Inter2-conc from R-Inter-conc using R-Struct because $P|P' \equiv P'|P$ for any processes P and P'. The remaining rules can be interpreted in the same manner. For details we refer to [26]. Although this semantics is simpler than the semantics used in [5], all language elements can be encoded.

2.3 Verification of ePASS-IoS Process Models

In this section we firstly give an outline of the algorithm and explain each step in more detail afterwards. The algorithm is roughly processing the following steps:

- Translate the graphical process models to a π-calculus representation. We explained this step for all language elements of ePASS-IoS in [5]. In fig. (2b) a simple example is illustrated.
- Infer a LTS by exploiting the SOS rules.
- Identify states which have no exit transitions. These states are deadlocks.
- Ignore the final legal deadlock, determine the shortest paths to all possible illegal deadlocks and select the shortest path of all of them.
- Determine all involved subjects and their actions involving on the path route.
- Give feedback to the UI and label the involved subjects and actions with a bold border in case the deadlock is illegal.

Inferring Labeled Transition Systems. The analysis of deadlocks is done on the transition system of the respective π-calculus terms. To obtain such a transition system, an ePASS process is firstly translated into an internal representation of an π-calculus expression. Afterwards, a transition system is inferred from this expression. Figure (2) shows the method exemplarily. In figure (2a) a simplified auction business process is shown. The bidder subject (S1) performs an internal action at first, then sends the payment and expects the delivery afterwards. The seller subject (S2) performs an internal action too, waits for a payment and then sends off the delivery. This ePASS process is translated into the π-calculus expression depicted in figure (2b). Thereby all "Do something" actions are translated into τ operators. The translation of Send actions leads to an overlined channel following the grammar: *<sender subject name>-<message type>-<receiver subject name>*. The channels for receive actions follow the same grammar except not using overlined channels identifiers. The π expression is translated to the transition system depicted in figure (2c) in accordance to the following definition.

Definition 2 (Labeled Transition System of a Process). *Let P be a π-calculus process. The transition system of P consists of:*

- *the set S of states which contains P itself and all processes which are reachable from P via transitions,* and
- *the transition relation $\longrightarrow$ between processes in S, which is specified by derivation rules given in table 2.*

The start state of the process P is referred to the root node of the **L**abeled **T**ransition **S**ystem. Every edge to successor states will go out of this node. Both of the π processes $S1$ and $S2$ start with a τ operation. Therefore, the R-Par rule of table 2 combined with the R-Tau rule leads to the first node of the "left" branch of the LTS. Either $S1$ will perform firstly the action followed by performing τ of $S2$ or the other way around will take place. The "right" branch of

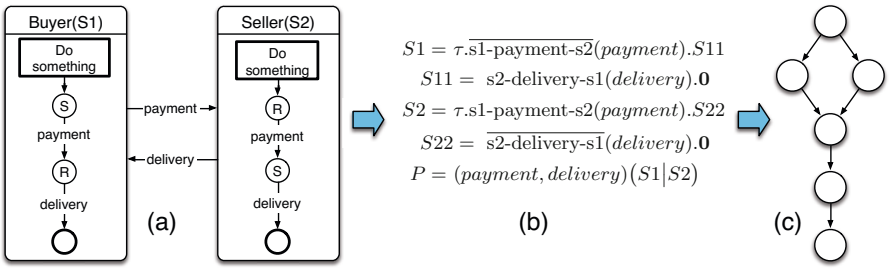

Fig. 2. The sequences of *S1* and *S2* (a) are translated to π expressions (b). By exploiting the SOS rules, defined in table (2), a LTS is inferred (c) in accordance to definition 2. The capacity of the subject input-pools are set to 0 in order to obtain a simple LTS.

the LTS can be inferred of by applying the R-Struct rule. Since the process P is restricted by the exchange of the messages *payment* and *delivery* , the message *payment* can be exchanged only between $S1$ and $S2$ in the next two steps. This system is used for the deadlock analysis. A deadlock is a state without outgoing transitions. The example transition system has only one deadlock state which is also the final state. It is called legal deadlock since we are modeling business process instances which have to be terminated at the end of their runtime. Every subject is equipped with a storage for incoming messages. This storage is called input pool and its data structure is a set of FIFO buffers. One FIFO buffer is needed for every receive exit condition which is a pair of (<Message-Type>, <Sender Subject>). The inputpool enables asynchronous communication and the multi set data structure provides a flexible access to the stored messages. It has a limited capacity which can be set in the editor. Without loss of generality, all FIFO buffers of a subject have the same capacity.

Different Cases of Deadlocks. When a deadlock was found, four different cases have to be distinguished.

All internal behaviors and all inputpool actors are terminated. In this case the process instance can always finish well.

Some internal behaviors are not terminated but all inputpool actors are terminated. The non terminated internal behaviors are in a receive state but no message will be send to them.

All internal behaviors are terminated and some inputpool actors not. In this case some messages left in the non terminated input pools. This can happen when either messages are send in a loop or when the internal behavior of the sender subject has one send transition more then the internal behavior of the receiver subject.

Some internal behaviors are terminated, others are still in the start state. When a process model includes a subject that only will be used optionally, this subject will stay in the start state.

The first case is a valid case. The second one is not valid, the cases 3 and 4 can be both and the verification system can not make this decision without input of the model designer. Therefore the system informs the user who can specify whether this is a valid case or not.

2.4 Implementation

A graphical editor was developed for the verification of the subject oriented process models. Fig. 3 and 4 show screenshots of the editor. The editor follows the same designing principles like the original Metasonic[19] editor but is using the ePASS-IoS elements and is extended by the check for interaction soundness. The implementation of the editor is based on the Graphical Editing Framework (GEF) [2] and on frameworks of the Metasonic AG. GEF is an open framework of the Eclipse environment and provides the opportunity to implement a graphical user interface for an existing data model. The Metasonic frameworks enabled the use of data binding and the pre-compiler AspectJ[1] RCP plug-ins were implemented which can be integrated into the Eclipse environment to use the ePASS editor.

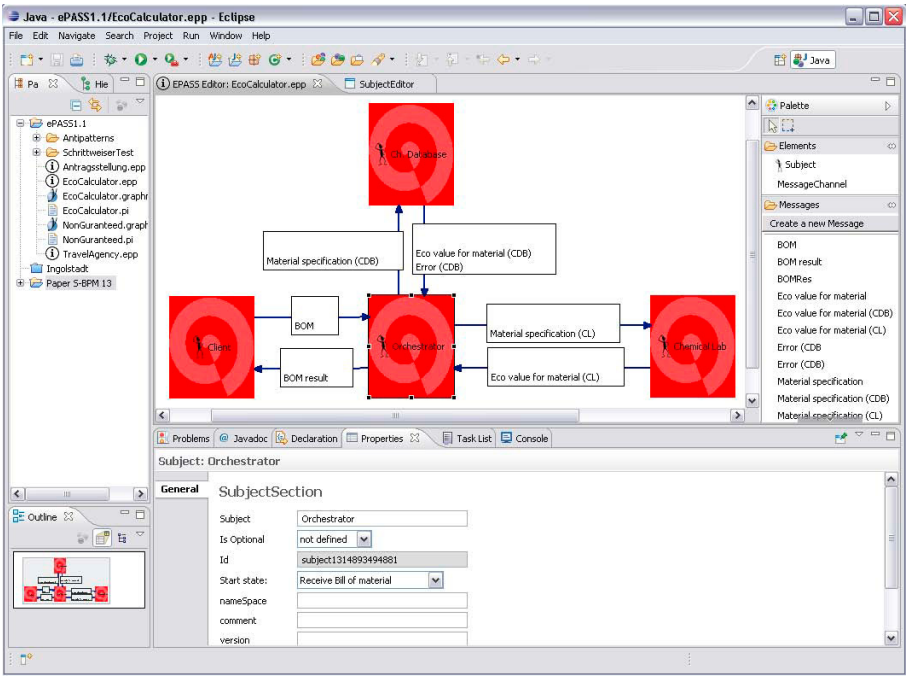

Fig. 3. Screenshots of the eclipse based editor, developed for the evaluation. The subject interaction view is shown with one of the test process models.

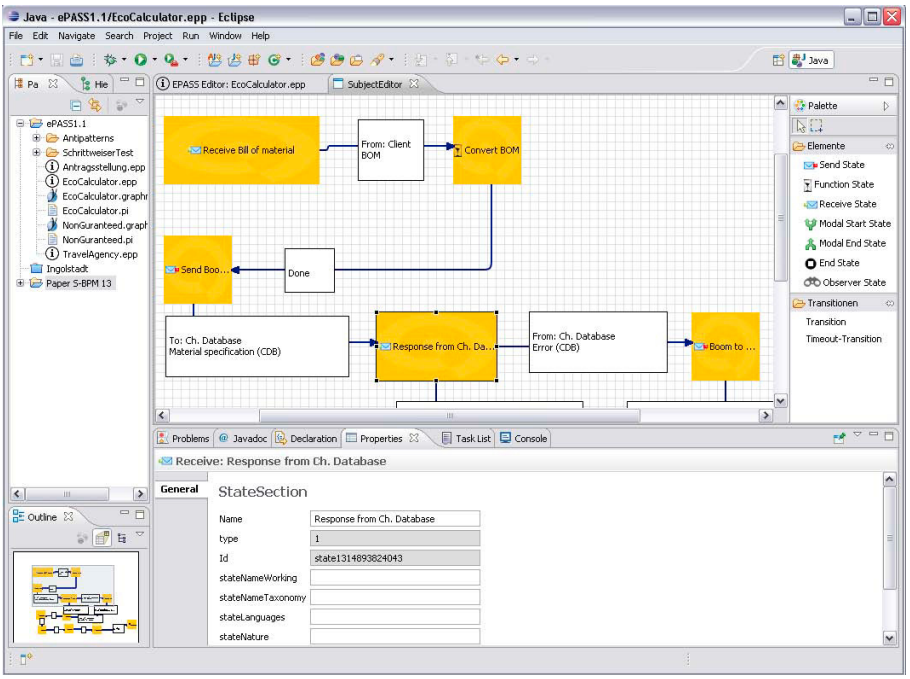

Fig. 4. Screenshots of the eclipse based editor, developed for the evaluation. An internal behavior of a subject is illustrated.

3 Evaluation

In [7] eight choreography patterns were identified. We have chosen these patterns for our evaluation because they are the most suitable ones for our purpose. The anti-pattern introduced in [4] tackle problems concerning compliance checking. They are not suitable for issues which can be arise by incompatible interaction behavior. Some of the patterns introduced in [3] are covered by anti-patterns described in [7]. We translated the choosen patterns including the examples in subject oriented models and verified them afterwards.

The input pool is the tool which enables asynchronous communication and has a limited capacity. Therefore a S-BPM process consisting of two subjects can run into an illegal deadlock in exactly three cases. Firstly, two participants are waiting for each other. Secondly, one participant has already terminated and the other one is still waiting for a message. Thirdly, the input pool limit of one participant is exceeded and the other one will send a message. The first two cases are particularly well represented by the contradicting sequence flow and the not-guaranteed termination anti-pattern. The last case is non of the 8 introduced anti-patterns but we evaluated that our approach detect this pattern as well.

3.1 AP1: Incomplete Sequence Flow

The first anti-pattern is called *Incomplete sequence flow* and is the most simple one. A path in the internal behavior of a subject is interrupted. This anti-pattern is already detectable by the check of structural soundness which is included in the Metasonic suite.

3.2 AP2: Contradicting Sequence Flow

A *contradicting sequence flow* means that the order of a send / receive action sequence is contradicted in two participants who are interacting with each other. An example for this anti-pattern is shown in figure 5. The bider will do the payment after the delivery arrived and the seller will deliver after the payment arrived. This leads to a classical deadlock situation. The final state of the model LTS is an illegal deadlock as shown in the right figure of (5). In the example each subject provider can only perform the internal action *Do something* and is locked in the receive state afterwards. Hence the LTS has exactly 4 states. Because of the non deterministic behavior either the Bidder or the Seller can perform the first action. When the Bidder performs the first action the path (0 -> 1 -> 3) is generated. Otherwise the path (0 -> 2 -> 3) of the LTS in figure (5) is generated.

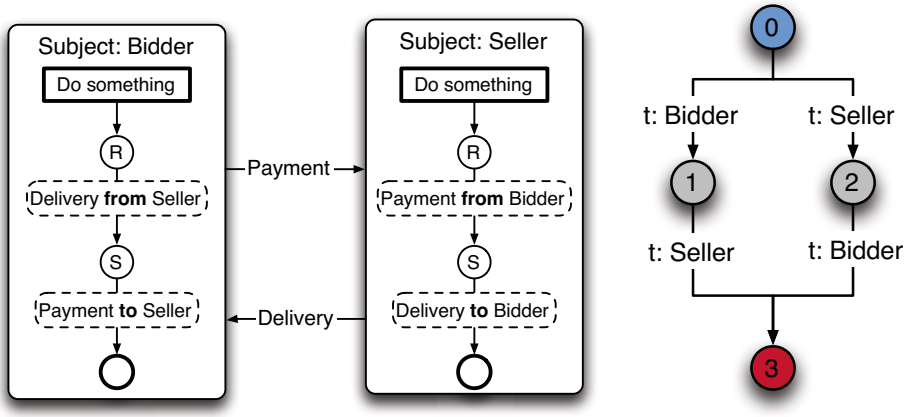

Fig. 5. Contradicting sequence flow. Each participant is waiting for each other what leads to an illegal deadlock (state 3). t:Bidder denotes that the Bidder performs an internal action τ.

3.3 AP3: Not-guaranteed Termination

An example for an *not-guaranteed termination* is depicted in figure 6. A bidder will process always an *Feedback* request / replay sequence but the *Auctioning Service* demands only sometimes one. When the *Auctioning Service* demands no feedback the *Bidder* is running into a deadlock. The concerning LTS includes besides the legal deadlock (state 8) the illegal deadlock (state 4)

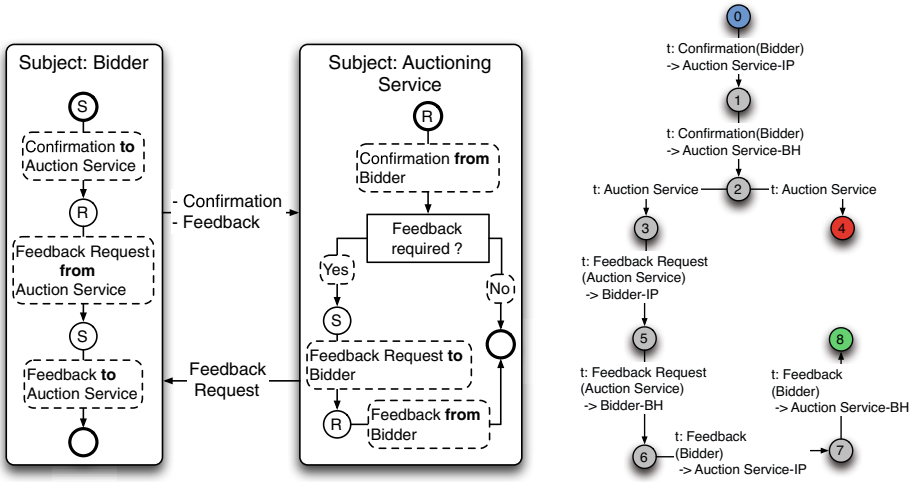

Fig. 6. A not guaranteed termination process snippet and its concerning LTS. The Bidder always expect a Feedback request but the Auctioning Service not.

3.4 AP4: Incompatible Branching Behavior

The next anti-pattern is called *incompatible branching behavior*. Internal behaviors of different subjects can contain same states with same transitions. If the decision made at these states are different, the control flow will take different branches and therefore the interaction behavior can become incompatible. This issue is depicted in figure 7 The two subject exchange the messages *Bank transfer* and *Credit card*. Both internal behaviors contain the action state *Payment* which has two successor transitions each. The exit conditions of the transitions are *Bank transfer* and *Credit card*. When the two subjects have different information about the payment options, the different exit conditions lead to different successor states. Since the successor states are *Send* and *Receive* actions, which will exchange the concerning messages, the interaction behavior is not compatible to each other.

When we apply the verification on these patterns, the verification will state that the model is not valid and that the labeled transition system contains two faulty end-states. They are colored red in figure 7 which depicts the inferred LTS of the process model. The LTS was created with the yed editor [29] which offers functions for automatic layouting. The green end state in figure 7 is the legal deadlock. In this state all system actors are terminated, namely the actors for the internal behavior and for the inputpool of each subject. The conditions have two different transitions and therefore four different exit condition combinations are possible. When the bidder and the seller make the same choice of of payment options, the successor paths will lead to the same end state, the legal deadlock state with number (15) in the figure. When the bidder choose *Credit card* and the seller choose *Bank transfer* the paths will lead to one red deadlock state.

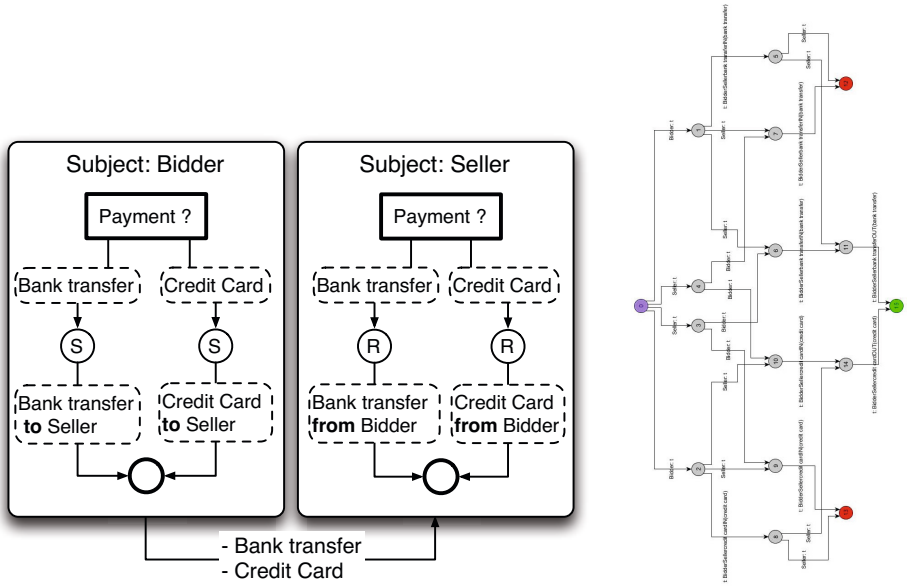

Fig. 7. Incompatible branching behavior of the subjects *Bidder* and *Seller*. The execution of the process model starts at the purple state in the LTS and can lead to one legal deadlock (green) and two illegal deadlocks (red).

The other way around the system will end in the other deadlock state. Another variant of this anti-pattern contains loops but the result is the same.

3.5 AP5: Impossible Data-Based Decisions

Sometimes process model designers put decisions in the model which are supposed to work on data not be available. These *impossible data-based decisions* are not detectable by the structural soundness check and only in certain cases by the formal verification for interaction soundness. The formal verification works on an abstract level and takes every internal action as a τ action. Therefore impossible data-based decisions can only be detected if they have an impact on later interaction actions. In these cases the anti-pattern is treated in the same way like the incompatible branching behavior pattern. The other cases effects only the participant where the error occurs and can not block the hole process. In figure (8) an example is outlined where it is assumed that the *Bidder* works on not available data.

3.6 AP6 Optional Participation

Not all of the including subjects have to participate in every process instance. 4 eye principles or fallback solutions are examples for this and the concerning pattern is called *optional participation*. When a subject is not being used, this

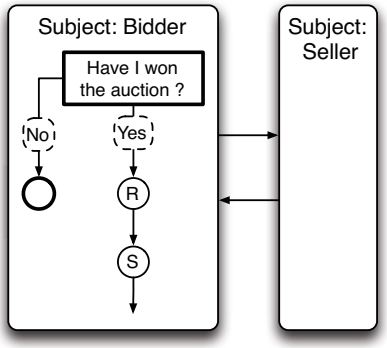

Fig. 8. Outline of the impossible data-based decisions anti-pattern. The *Bidder* can not know whether the auction is won. These decisions lead most often to incorrect interaction behavior.

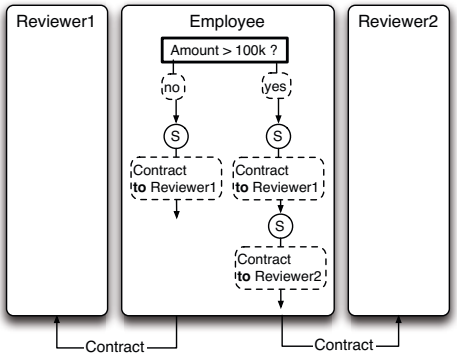

Fig. 9. Outline of the optional participation anti-pattern. When the contract is not send to *Reviewer2*, the concerning internal behavior stays at the start state. This leads to a deadlock.

subject will stay in its start state or in one of its first receive states. This is a deadlock situation: The verification system detects the deadlock and throws an error message on the UI first. The option to ignore this deadlock is included in the message dialog. The process designer can inspect the issue and decide whether it is a wanted or not wanted situation. If the process designer labeled it as wanted the verification system is not throwing the error message again.

The 4 eye principle process in figure 9 is a good example for optional participation. An insurance has to cover a damage and when the amount of loss exceeds a certain threshold, a second reviewer is needed. In other cases the subject *Reviewer2* is still waiting for an application and the overall process can not terminate.

3.7 AP7: Uni-lateral Sequentialization

Uni-lateral sequentialization is another anti-pattern. Because of the modal split / join operator pair, sequences can be processed in parallel. The subject *Seller* in figure 10 is processing a part of the interaction actions in parallel. The behavior is more permissive than it would be in the case that all interaction actions are ordered in only one sequence. This leads not to an error in ePASS modeled processes, since the subjects store received messages in their input pools. Although, the processing of certain tasks can be delayed. If incompatible interaction behavior occurs as aftereffect, our method will detect this.

3.8 AP8: Mixed Choices

One of the most complex anti-pattern is the *mixed choices* pattern. It consist of a branching of a send and a receive action. The send action can as long take

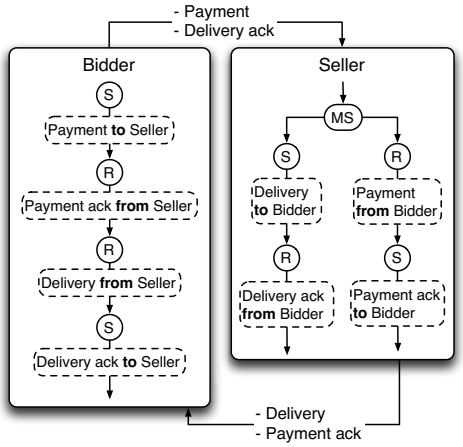

Fig. 10. The *uni-lateral sequentialization* anti-pattern does not lead directly to errors. It just states that one internal behavior is more permissive than the other one. If incompatible interaction behavior occurs as after-effect, our method will detect this.

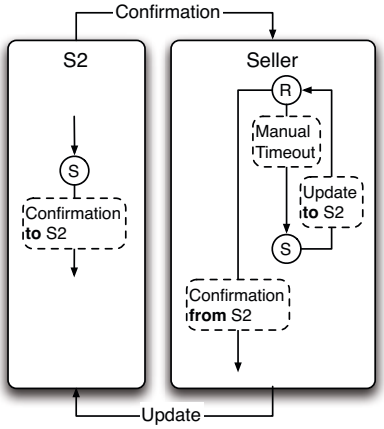

Fig. 11. The *mixed choices* anti-pattern is hard to model correct. When it leads to a left message in the input pool, the verification method would find it.

place as the receive action did not receive a message. The figure (11) gives an example for that. The seller can update its setting until the confirmation message is received. The solution for this kind of problem is not obvious and according to [7] even experienced modeler make errors describing such situations. While the seller is in the branch of sending update information, the *Confirmation* message could be received. Due to this overlapping, the *Update* message could be left in the input pool of subject *S2* which would be detected by our verification method as explained in section (2.4).

3.9 Performance of the Verification Method

Next to the detection of the anti-patterns the solution is supposed to be that fast, that it does not slow down the work of the model designer to much. To check whether our performance requirements are met, we applied the algorithm to ten different process models. Three of them were delivered with the S-BPM Suite from Metasonic [19] and serve as example process models for customers. One is a process modeled as use case in the Theseus / TEXO project [24]. Since these process models could be verified in less than a second, we modeled 6 further fictional process models. The fictional process models were developed such a way that LTSs with a large amount of states could be inferred easily . The tests were run on a notebook with an Intel Core 2 Duo microprocessor and with 4 GB RAM. The operating system was Windows 7. The results are listed in table 3. Nine of ten models could be verified in less then 15 seconds and eight of ten in

Table 3. Verification time of different process models. F1 to F6 are fictional process models. They were developed by using a high number of modal split and modal join actions.The action sequences between these actions can be executed in parallel and therefor a large number ofLTS states could be inferred. Only the verification time of F6 exceed the required limit of 15 seconds.

Process	Subjects	LTS states	Verification time $[s]$
Eco Calculator	4	52	< 1
Application Process	3	22	< 1
Banking process	5	105	< 1
Ordering Process	4	28	< 1
F1	2	20	< 1
F2	3	108	< 1
F3	4	271	< 1
F4	5	559	< 1
F5	6	4072	8
F6	7	12223	41

less than one second. All of the practice related process models ($F1$-$F4$) could be verified in less than a second. Although the verification of $F6$ took 41 seconds, which is almost 3 times higher than the required limit, in summary the results can be scored as good. Nevertheless the algorithms should be improved since we could not verify real world processes.

3.10 User Support for Error Correction

We required not to use external verification tools but integrate verification algorithms into the tool. The reason for that is to get the ability providing better support for the user to correct errors. We developed two mechanisms to achieve this goal. The first mechanism is to visualize possible error sources graphically. Therefore, the shortest path from the start state to the deadlock state is determined. This path delivers information about the involved subjects and the involved actions of the subjects. Hence it is possible to label these elements. We have labeled the involved subjects and the involved internal actions with a big blue frame. The user can easily see which elements are involved and therefore can better trace the error sources. In some cases many subjects can be involved on the path from the start state to a deadlock state. The ability to detect errors suffer from this fact and so a second mechanism has been introduced to find error sources. It is called stepwise check and gives the ability to check only parts of the process. The user can select subjects which are supposed to verify. Instead of the remaining subjects an open environment is used. This environment is an ideal interaction partner. It receives all messages the subjects will send and delivers necessary messages when the subjects requires them. If the verification method detect errors, the error sources can be localized within the subjects verified. This way the process designer is able to find errors systematically.

4 Related Work

In [25] a solution for checking interaction soundness for service orchestrations is introduced. The authors introduce an graphical example but have to write the concerning π-calculus terms manually, since the BPMN related, graphical notation does not allow an automatic transformation. To evaluate the method the Mobility Workbench (MWB) is used[27,28]. The tool takes a π-calculus expression as input and outputs a binary result in this case. Either the check is positive or negative. Therefore they can not use this result to analyze the error sources. Further more, the tool is not optimized for this purpose in that sense, that certain loop constructs are not detectable. In [13] a BPMN extension is introduced together with a graphical editor. This editor enables the automatic transformation to π-calculus terms. The weaknesses of the verification method itself are still unsolved.

In [16,17] Lohmann et al. transform BPMN and BPEL models to *Open Workflow Nets*. The *Open Workflow Nets* are verified with a model checker tool and exploiting a graph edit distance and deadlocks are supposed to solve automatically. The algorithm is not able to discover in all cases which participant behavior causes the error. No graphical support is provided for the model designer to solve the cause of the error in that cases it can not be done automatically.

In the work of Deng et al.[8] services are described by the Web Service Definition Language (WSDL) and were transformed to π-calculus expressions. They also propose to use the MWB to verify the expressions though without giving details how the results could be used to detect the reasons for errors.

A further approach is introduced in [15]. Business process are described with BPEL and the Business Property Specification Language (BPSL). The BPEL process were transformed to π-calculus expressions and were inferred into finite state machines. Linear temporal logic (LTL) forms were generated from the BPSL expression. Both kinds of expressions were given to a compliance checking framework called OPAL. Again, the outputs are either the check failed or the check succeeded.

In [4,18] approaches for compliance checking are featured. These approaches give visual feedback to the model designer but can not be used for compositions of participants. The capacity and the structure of the technique for asynchronous communication must be taken into account. Both approaches fail concerning this point.

5 Summary and Conclusion

Finding deadlocks in distributed systems and resolving them are two very challenging tasks. These challenges are well known and many different formalisms, algorithms and tools have been developed over the years to tackle the associated problems. Nevertheless it is not clear which solution is suitable for finding deadlocks in S-BPM processes and resolving them. In this paper it has been shown how ePASS-IoS 1.1 process models can be verified for interaction soundness by

using the π-calculus as formal foundation. An graphical editor was developed and the verification algorithms were integrated. Further more, two mechanisms to support the user finding the reasons for errors are integrated. We modeled the eight choreography anti-patterns in ePASS-IoS and investigated which of them can be detected. The *incomplete sequence flow* anti-pattern can already be detected by the structural soundness check which is integrated in the Metasonic Suite. Five of them can always be detected namely *contradicting sequence flow*, *not-guaranteed termination*, *incompatible branching behavior*, *impossible data-based decisions* and *optional participation*. The remaining two *Uni-lateral sequentialization* and *mixed choices* can be detected when they lead to incompatible interaction behavior. Further more, we evaluated that the performance is fast enough to use the method in practice, although the algorithms are simple.

For future work, we currently plan the following:

- Extend ePASS IoS and formal semantics to support multi subjects.
- Optimize the algorithms to obtain even faster results to be able to verify also very complex processes.
- Combine the verification method with Abstract State Machines.

Acknowledgments. We thank Kai Mehringskötter for his great work developing the graphical editor and implementing the verification method. We thank Albert Fleischmann for fruitful discussions and valuable comments. The work presented in this paper was performed in the context of the Software-Cluster project SINNODIUM (www.software-cluster.org). It was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. "01IC10S01" | "01|C10S05". The authors assume responsibility for the content.

References

1. AspectJ, `http://www.eclipse.org/aspectj/` (last accessed on January 15, 2014)
2. Graphical Editing Framework, `http://www.eclipse.org/gef` (last accessed on January 15, 2014)
3. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service interaction: Patterns, formalization, and analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009), `http://dx.doi.org/10.1007/978-3-642-01918-0_2`
4. Awad, A., Weske, M.: Visualization of Compliance Violation in Business Process Models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 182–193. Springer, Heidelberg (2010)
5. Borgert, S., Steinmetz, J., Mühlhäuser, M.: ePASS-IoS 1.1: Enabling Inter-enterprise Business Process Modeling by S-BPM and the Internet of Services Concept. In: Schmidt, W. (ed.) S-BPM ONE 2011. CCIS, vol. 213, pp. 190–211. Springer, Heidelberg (2011)
6. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology 50(2), 122–127 (2008)
7. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Information Systems 36(2), 292–312 (2011), `http://linkinghub.elsevier.com/retrieve/pii/S0306437910000591`

8. Deng, S., Wu, Z., Zhou, M., Li, Y., Wu, J.: Modeling Service Compatibility with Pi-calculus for Choreography. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 26–39. Springer, Heidelberg (2006)
9. Fleischmann, A.: PASS - A Technique for Specifying Communication Protocols. In: Proceedings of the IFIP WG6.1 Seventh International Conference on Protocol Specification, Testing and Verification VII, pp. 61–76. North-Holland Publishing Co, Amsterdam, http://portal.acm.org/citation.cfm?id=645831.670083
10. Fleischmann, A.: Distributed Systems: Software Design and Implementation. Springer, Berlin (1994)
11. Fleischmann, A.: What Is S-BPM? In: Buchwald, H., Fleischmann, A., Seese, D., Stary, C. (eds.) S-BPM ONE 2009. CCIS, vol. 85, pp. 85–106. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-15915-2_7
12. Fleischmann, A., Lippe, S., Meyer, N., Stary, C.: Coherent Task Modeling and Execution Based on Subject-Oriented Representations. In: England, D., Palanque, P., Vanderdonckt, J., Wild, P.J. (eds.) TAMODIA 2009. LNCS, vol. 5963, pp. 78–91. Springer, Heidelberg (2010)
13. Freund, J., Rcker, B., Henninger, T.: Praxishandbuch BPMN. Hanser (2010), http://books.google.com/books?id=bw9YPgAACAAJ
14. Keller, R.M.: Formal Verification of Parallel Programs. Communications of the ACM 19(7), 384 (1976), http://portal.acm.org/citation.cfm?id=360248.360251
15. Liu, Y., Müller, S., Xu, K.: A static Compliance-Checking Framework for Business Process Models. IBM Systems Journal 46(2), 335–361 (2007), http://dx.doi.org/10.1147/sj.462.0335
16. Lohmann, N.: Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer, Heidelberg (2008)
17. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
18. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., Dadam, P.: Seaflows toolset compliance verification made easy for process-aware information systems. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 76–91. Springer, Heidelberg (2011)
19. Metasonic: Metasonic Suite (2014), http://www.metasonic.de/ (last accessed on January 15, 2014)
20. Milner, R.: Communicating and mobile systems: The π-calculus. Cambridge University Press (1999)
21. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
22. OMG: Business Process Modeling Notation. 2.0 edn. (2012), http://www.omg.org/spec/BPMN/2.0/ (last accessed on January 15, 2014)
23. Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, University of Aarhus (1981)
24. Project of the German Federal Ministry of Economy and Technology: TEXO Infrastructure for Web-based services (2012), http://theseus.pt-dlr.de/en/914.php (last accessed on January 15, 2014)
25. Puhlmann, F., Weske, M.: Interaction Soundness for Service Orchestrations. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)

26. Sangiorgi, D., Walker, D.: The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press (2003)
27. Uppsala Universites, Department of Information Technology: The Mobility Workbench (2006), http://www.it.uu.se/research/group/mobility/mwb (last accessed on January 15, 2014)
28. Victor, B., Moller, F.: The Mobility Workbench - a tool for the π-calculus. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 428–440. Springer, Heidelberg (1994)
29. yWorks: yED Graph Editor (2012), http://www.yworks.com/ (last accessed on January 15, 2014)