Analyzing BPEL4Chor: Verification and Participant Synthesis

Niels Lohmann[1], Oliver Kopp[2], Frank Leymann[2], and Wolfgang Reisig[1]

[1] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
{nlohmann, reisig}@informatik.hu-berlin.de
[2] Institute of Architecture of Application Systems, University of Stuttgart, Germany
Universitätsstraße 38, 70569 Stuttgart, Germany
{kopp, leymann}@iaas.uni-stuttgart.de

Abstract. Choreographies offer means to capture global interactions between business processes of different partners. BPEL4Chor has been introduced to describe these interactions using BPEL. Currently, there are no formal methods available to verify BPEL4Chor choreographies. In this paper, we present how BPEL4Chor choreographies can be verified using Petri nets. A case study undermines that our verification techniques scale. Additionally, we show how the verification techniques can be used to generate a stub process for a partner taking part in a choreography. This is especially useful when the behavior of one participant is intended to follow the corresponding requirements of the other participants. Thus, the missing participant behavior can be generated and the error-prone design of that participant can be skipped.

Key words: BPEL4Chor, choreography, partner generation, Petri nets, service-oriented analysis and design

1 Introduction

The Web Services Business Process Execution Language (WS-BPEL or BPEL for short, [1]) is the de facto standard to describe executable business processes as orchestrations of Web services. A *choreography* describes the interaction of several processes from a global perspective. In particular, it defines the order in which processes exchange messages. BPEL4Chor [2] is a choreography language based on BPEL. Each participant is associated with a *participant behavior description* (PBD) that describes the participant's behavior using abstract BPEL. The interconnection between the activities of different PBDs is formed by message links.

In this paper, we show how an existing tool chain [3,4] can be extended to analyze a BPEL4Chor choreography (Fig. 1). By mapping BPEL4Chor to Petri nets, we also provide a formal model for BPEL4Chor.

If two business partners agree on a choreography, but need a third business partner to achieve their goal, they also have to specify the behavior of the

Niels Lohmann, Oliver Kopp, Frank Leymann, and Wolfgang Reisig. **Analyzing BPEL4Chor: Verification and participant synthesis**. In Marlon Dumas and Reiko Heckel, editors, *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28–29, 2007, Proceedings*, Lecture Notes in Computer Science. Springer-Verlag, September 2007. Note: accepted.

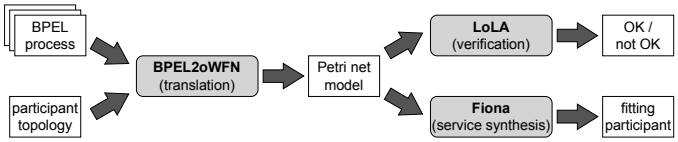


Fig. 1. Proposed tool chain to analyze BPEL4Chor choreographies.

third party. We show how the behavior of the third party can be derived from existing participants in a choreography. The current algorithms assure deadlock-freedom for the synthesized participant if such a participant exists. We are aware that there are other possibilities for defining "proper interaction". Nevertheless, deadlock-free communication will certainly be part of any more sophisticated correctness definition, so the presented approach can be seen as a step towards a more sophisticated solution.

Section 2 introduces BPEL4Chor and open workflow nets (oWFNs), which are used to capture the semantics of BPEL4Chor. After presenting in Sect. 3 how BPEL4Chor can be translated into oWFNs, Sect. 4 shows how a BPEL4Chor choreography can be analyzed theoretically. Section 5 puts that analysis into practice and shows how the proposed tool chain is used to analyze a BPEL4Chor choreography and that it scales up to 1,000 participants. Finally, Sect. 6 concludes, compares the presented work with related work, and describes future research directions.

2 Background and Motivation

A choreography described by BPEL4Chor consists of (i) the participant topology, (ii) the participant behavior descriptions, and (iii) the participant groundings (cf. Fig. 2 and [2]). The participant topology lists all participants taking part in the choreography and all message links connecting activities of different participants. A message link states that a message is sent from the source of the message link to its target. Every participant has a certain type. For each participant type, a participant behavior description (PBD) defined in BPEL is given. In this description, port types and operations are omitted and thus the dependency on interface specifications such as WSDL [5] is removed. If the choreography has to be executed, every target of a message link has to be grounded to a WSDL operation so that the other participants can use the offered operation. This grounding is done after the choreography design itself, which enables choreography specification reuse. Since BPEL is used to specify the behavior of every participant, the development of executable BPEL processes following this behavior can be done by using the PBD of a participant as a basis and adding missing information. Other languages can be used to provide implementations of local behavior, but using BPEL is a seamless choice based on BPEL4Chor.

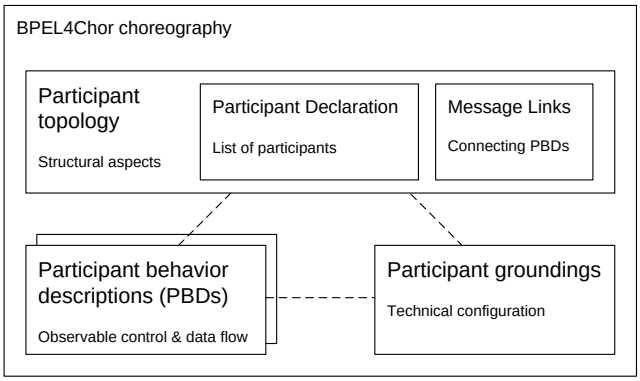


Fig. 2. BPEL4Chor artifacts. ([2])

A choreography always describes the behavior of all participants. Thus, a *closed world* is assumed. Refer to the booking scenario in Fig. 3. A traveler requests booking of a flight at a travel agency. The travel agency requests a price quote from every airline in a set of airlines. The cheapest airline is selected and the tickets are ordered there. The airline replies with a confirmation and sends an electronic ticket directly to the traveler. There is no message going to an undefined participant. The observable behavior of all participants is specified. Note that BPMN [6] is used for visualization only. The choreography itself is specified using BPEL4Chor.

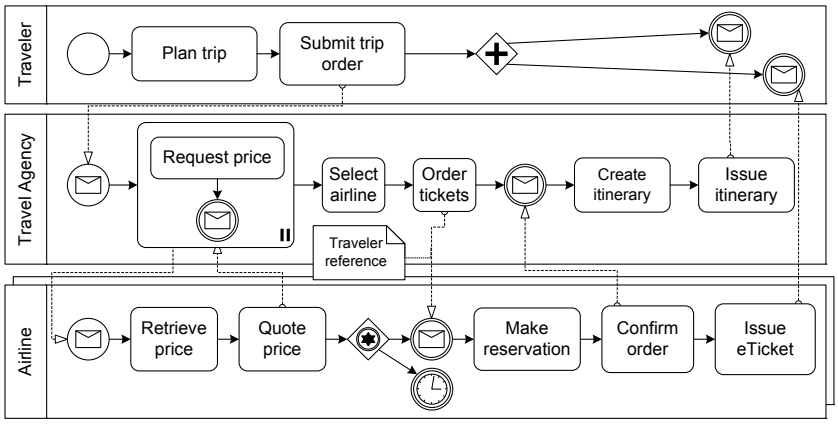


Fig. 3. Choreography of a Booking Scenario. ([2])

2.1 Open Workflow Nets

Open workflow nets (oWFNs) [7] are a special class of Petri nets. They generalize classical workflow nets [8] by introducing an interface for asynchronous message passing. Intuitively, an oWFN is a Petri net together with (i) an *interface*, consisting of input and output places, (ii) an initial marking m_0, and (iii) a set Ω of distinguished *final markings*. Final markings represent desired final states of the net and help to distinguish desired final states from unwanted deadlocks. Throughout this paper, we use the term 'deadlock' for a nonfinal marking which does not enable a transition (i. e., an unwanted blocking of the net). Figure 4 shows an oWFN modeling the traveler service of the choreography depicted in Fig. 3.

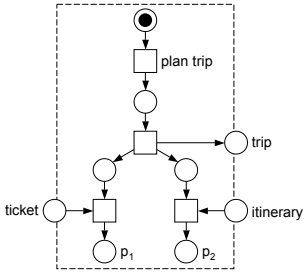


Fig. 4. An oWFN modeling the traveler service. The interface consists of $P_{\mathsf{in}} = \{\mathsf{ticket}, \mathsf{itinerary}\}$ and $P_{\mathsf{out}} = \{\mathsf{trip}\}$, depicted on the dashed frame. The traveler first plans the trip and then sends an order. Then, he concurrently receives a ticket and an itinerary. The set of final markings $\Omega = \{[\mathsf{p}_1, \mathsf{p}_2]\}$ consists of the single marking with one token on place p_1 and on p_2.

The interplay of two oWFNs N and M is represented by their *composition*, denoted by $N \oplus M$. Thereby, we demand that the nets only share input and output places such that for some input places of N exist corresponding output places of M, and vice versa. The oWFN $N \oplus M$ can then be constructed by merging joint places and merging the initial and final markings. Merged places become internal to $N \oplus M$. Due to the closed world assumption in BPEL4Chor, the composition of all oWFNs modeling services of a choreography results in a *closed* oWFN; that is, an oWFN with empty interface.

oWFNs provide a simple but formal foundation to model services and their interaction. They allow — like common Petri nets — for diverse analysis methods of computer-aided verification. The explicit modeling of the interface further allows to analyze the interaction behavior of a service [3,4]. An important property of an oWFN is whether it is possible to communicate deadlock-freely with it. An oWFN N is called *controllable*, if there exists an oWFN M such that $N \oplus M$ is free of deadlocks. Like the soundness property for workflow nets, controllability [9] can be regarded as a minimal correctness criterion for communicating services. Obviously, the net depicted in Fig. 4 is controllable.

2.2 Petri Net Semantics for BPEL

The BPEL [1] language provides an operational semantics defining the behavior of each language construct and the behavior of composites of constructs. To

formally verify BPEL processes, a formal semantics is needed. Therefore, a lot of work has been conducted to define a formal semantics for the behavior of BPEL processes. The approaches cover many formalisms such as Petri nets, abstract state machines (ASMs), finite state machines, process algebras, etc. (see [10] for an overview).

The translation of a BPEL process intro a Petri net model is guided by the syntax of BPEL. In BPEL, a process is built by plugging instances of language constructs together. Accordingly, each construct of the language is translated separately into a Petri net. Such a net forms a *pattern* of the respective BPEL construct. Each pattern has an interface for joining it with other patterns as is done with BPEL constructs. Also, patterns capturing BPEL's structured activities may carry any number of inner patterns as its equivalent in BPEL can do. The collection of patterns forms the *Petri net semantics* for BPEL. While the original semantics in [11] is feature complete (i. e., capturing both the standard as well as the exceptional behavior of a BPEL process), we only consider the positive control flow in this paper to ease the presentation. The presented approach can, however, be canonically enhanced to also model fault, compensation and exception handling of the participating BPEL processes.

3 Translating BPEL4Chor Choreographies into Petri Nets

To translate a BPEL4Chor choreography into a Petri net model, we extend the translation approach presented in [3]. Basically, the translation is enhanced to support *composition* and *instantiation*.

Composition The tool chain presented in [3] is limited to the translation of a single process into a Petri net model. To translate BPEL4Chor, we translate the participating BPEL processes one by one and compose the resulting oWFNs. The information how input and output places of different processes are merged can be derived from the participant topology. As the composition of oWFNs is associative, the order of composition is not important. Furthermore, the resulting nets can be composed incrementally. Therefore, at most two nets have to be kept in memory during the translation process. Finally, structural reduction techniques can be applied already during the composition process. Not only the final composition, also the intermediate oWFNs can be reduced. This interleaving of structural reduction and composition does not only allow smaller nets, but also may speed up the translation process as the size of the composition grows more slowly.

Instantiation The translation process is, however, not restricted to choreographies in which each process is instantiated just once. For instance, the choreography example presented in Fig. 3 models a choreography that communicates with a *set* of airlines. Again, the participant topology holds the necessary information about which process has to be instantiated. Admittedly, the topology does not

provide the number of instances of each participant. We therefore demand an upper bound of instances to be specified for each participant set. While this upper bound may not be necessary when BPEL4Chor is just a means to *describe* choreographies, its definition is reasonable when such a choreography should be *analyzed*.

To introduce instantiation to the translation process, the following scenarios are possible:

i. message exchange between two uninstantiated participants (e. g., the trip order sent by the traveler to the agency),
ii. message exchange between an uninstantiated participant and one particular instantiated participant (e. g., the price request sent by the agency to an airline instance),
iii. message exchange between an uninstantiated participant and an arbitrary chosen instantiated participant (e. g., the e-ticket sent by the selected airline to the traveler), and
iv. message exchange between two instantiated participants (not present in our example choreography).

For an example of these scenarios, consider the BPEL code snippet of the agency process depicted in Fig. 5(a). For two airline instances, the resulting subnet is depicted in Fig. 5(b). The message trip sent by traveler to the agency is an example of the first scenario, as both services (traveler and agency) are uninstantiated. Therefore, the receipt of the trip message is modeled by a single transition, namely t_1. The price request sent to and the corresponding price quotes received from the airline instances are examples for the second scenario. Therefore, the communicating transitions (t_2–t_5) and the connected interface places (price.1, price.2, quote.1, and quote.2) are instantiated. The order sent to only one airline instance is an example for the third scenario.

Translating the example choreography The presented translation approach was implemented in our compiler BPEL2oWFN[3]. BPEL2oWFN enables us to translate real-world BPEL choreographies into Petri net models. We translated the example choreography with five airline instances into a Petri net. The resulting net has 103 places and 81 transitions. Structural reduction simplified the net to 63 places and 41 transitions. The final marking of the composition is constructed canonically: it consists of the single state in which all participating services have completed faultlessly.

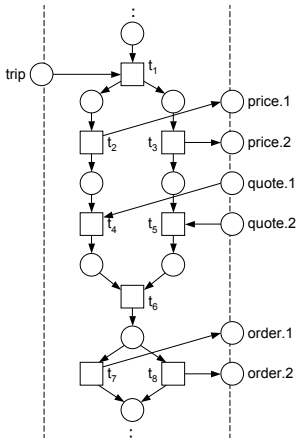
4 Analyzing BPEL4Chor Choreographies

In this section, we show how to analyze BPEL4Chor choreographies using Petri net models. We distinguish two analysis approaches: analysis of *closed* choreography

[3] BPEL2oWFN is available at `www.gnu.org/software/bpel2owfn`.

```
<receive wsu:id="ReceiveTripOrder" />
<forEach wsu:id="fe_RequestPrice" parallel="yes">
  <scope>
    <sequence>
      <invoke wsu:id="RequestPrice" />
      <receive wsu:id="ReceiveQuote" />
    </sequence>
  </scope>
</forEach>
<opaqueActivity name="SelectAirline" />
<invoke wsu:id="OrderTickets" />
```

(a) code snippet of the agency process



(b) resulting subnet of the agency oWFN

Fig. 5. Example for the instantiation of transitions and interface places. The BPEL process of the agency (a) is translated into an oWFN (b).

models and analysis of *open* choreography models. A closed choreography model (i. e., an oWFN with empty interface) can be analyzed in isolation and can be used to verify properties of a complete choreography. For example, deadlock-freedom or the absence of unwanted communication scenarios can be proven before the actual implementation and deployment of the participant services. In contrast, an open choreography model (i. e., an oWFN with nonempty interface) can be used during the design of the overall choreography. A choreography in which one participating service is missing can, for instance, be completed by synthesizing the missing participant service. This synthesized service is then guaranteed to participate deadlock-freely with the other participants.

4.1 Analyzing Closed Choreographies

Due to the closed-world assumption of BPEL4Chor, the resulting Petri net model of a completely specified BPEL4Chor choreography is a closed system; that is, a Petri net with empty interface. During the translation, each interface place of an intermediate oWFN, is merged with a corresponding interface place of another intermediate oWFN. Closed systems do not have an environment and thus their state space can be calculated and analyzed without considering the environment of the system. As Petri nets offer a broad variety of analysis methods, a lot of interesting properties can be investigated:

- Is the choreography free of deadlocks and livelocks? Will each participating service eventually reach a final state?
- Will a certain activity of a participant be executed? Does there exist a state in which more than one message is pending on a communication channel?

- What is the minimal/maximal number of messages to be sent to reach a final state of the choreography?
- Will a participant always receive an answer? Can a participant enforce the receipt of a certain message?

These questions can be formulated in terms of reachability or temporal logic properties and be checked using existing model checking tools.

Analyzing the example choreography We analyzed the Petri net model of Sect. 3 with the Petri net verification tool LoLA [12], a state-of-the-art model checker which implements several state space reduction techniques. The unreduced state space consists of 3,843 states. The Petri net model contains an unwanted deadlock. We could map this deadlocking state of the model back to the participating services with the help of a *witness path*. A witness path is a transition sequence leading from the initial to the dead marking. The deadlock occurs in the choreography, when the agency's choice for an airline takes too much time, or when the message sent to the chosen airline is delayed. In this case, the timeout (i. e., the `onAlarm` branch) of all participating airlines ends their instances and the agency deadlocks waiting for a confirmation message from the chosen airline.

Correcting the example choreography There are many ways to correct the deadlocking choreography. A straightforward attempt would be to replace the airline service's timeout by a message sent by the agency. This would, however, add an unrealistic dependency between the agency and the airline. To this end, we decided to keep the timeout, but at the same time ensure a response of the airline service even when a ticket order is received after the timeout.

Hence, we changed the choreography as follows (cf. gray shapes in Fig. 6). The airline's behavior does not change if the agency's ticket order is received before the timeout occurred and if the timeout occurs, the airline service's instance still terminates. However, a new branch was added to the airline: this branch models the situation in which the agency's ticket order is received after the timeout. In this case, a new instance of the airline service is created which rejects the ticket order. The services of the agency and the traveler are adjusted to handle this rejection.

Analyzing the new example choreography We translated the new choreography with five airline instances into a Petri net model. Due to the newly introduced activities, its structure and its state space have grown. The (structurally reduced) net has 113 places and 97 transitions. The model has 3,812 states and does not contain deadlocks except final states. With the help of LoLA, we could also verify that the choreography's participating services do not livelock and will always reach a final state.

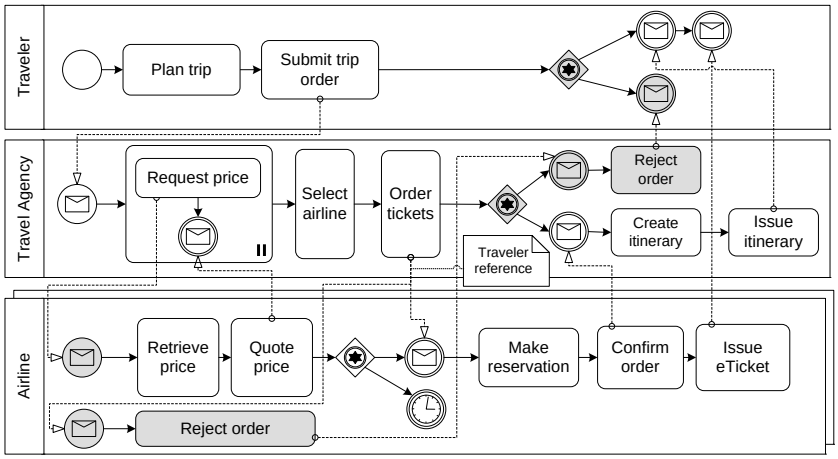


Fig. 6. Deadlock-free Choreography of a Booking Scenario. The two start events at the airline process denote a BPEL pick activity.

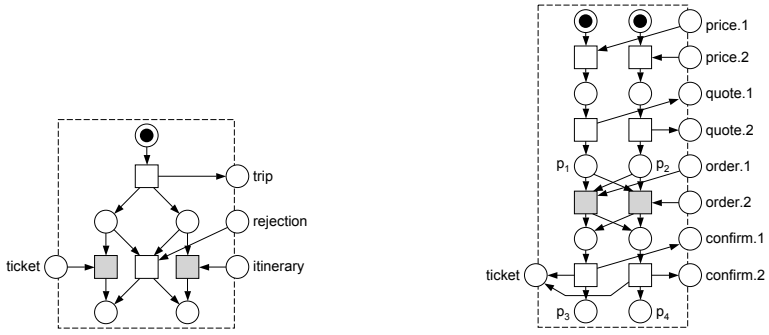
4.2 Analyzing Open Choreographies

While the analysis of closed choreographies may help to find design flaws in the interaction between all participating services, Petri net models may also support the *design* of choreographies. To this end, controllability (cf. Sect. 2.1) is an important property. In [3], we presented an algorithm to decide controllability of an oWFN constructively. This algorithm is implemented in the tool Fiona[4]. If a partner exists such that the composition is deadlock-free, it is automatically generated.

Let $N_1, \ldots, N_{k-1}$ be the oWFNs of the already known participant services of an open choreography. Their composition, $N_1 \oplus \ldots \oplus N_{k-1}$, is an oWFN with nonempty interface. If this net is controllable, then there exists an oWFN N_k such that $N_1 \oplus \ldots \oplus N_{k-1} \oplus N_k$ is deadlock-free. Thus, Fiona can be used to "complete" a given open choreography by synthesizing the model N_k of the missing participant.

Synthesizing a traveler service Consider again the fixed choreography of Fig. 6. If, for example, only the services of the agency and the airline were specified, the blueprint of a traveler service could be synthesized. If such a service exists (i. e., the composition of the existing services is controllable), it completes the choreography which is then deadlock-free by construction. To this end, the incomplete choreography is translated into an oWFN using BPEL2oWFN. This oWFN is then analyzed by Fiona. If the net is controllable, a service automaton modeling the behavior of a partner service is synthesized. This automaton can be translated into an oWFN, for example using the tool Petrify [13].

[4] Fiona is available at `www.informatik.hu-berlin.de/top/tools4bpel/fiona`.



(a) synthesized traveler (b) two synthesized airline instances

Fig. 7. Synthesized services. (a) A traveler service synthesized to fit in the new choreography. The gray transitions are concurrent, whereas in the choreography (cf. Fig. 6), the ticket is received *after* the itinerary. (b) Two synthesized airline instances to fit in the first choreography (cf. Fig. 3). The gray transitions synchronize the instances. The net has two final markings: $\Omega = \{[\mathsf{p}_2, \mathsf{p}_3], [\mathsf{p}_1, \mathsf{p}_4]\}$.

The synthesized oWFN of a traveler service that completes the choreography is depicted in Fig. 7(a). This traveler service slightly differs from the traveler service in the new choreography (cf. Fig. 6). Firstly, there exists no transition modeling the planning of the trip, because such a transition is internal (i. e., not communicating), but the service was synthesized based on the external behavior; that is, only the interaction of the service was taken into account. Secondly, the itinerary and the ticket can be received concurrently. This is due to the asynchronous communication model: messages can keep pending on the interface, so there is no order in which they have to be received. From this oWFN, an abstract BPEL process can be derived using existing approaches [14]. As this translation is out of scope of this paper, we do not present it here.

Limits of the partner synthesis The presented approach allows to synthesize a service that interacts deadlock-freely with the other participating services of the choreography if such a service exists; that is, if the open choreography is controllable. At present, it is, however, not possible to synthesize a *set* of services which complete a choreography.

As an example, consider again the first (deadlocking) choreography of Fig. 3. The choreography deadlocks because of the airline service's timeout mechanism. If we synthesize the airlines, the result will be a *single* oWFN modeling the behavior of *all* airline service's instances.

Figure 7(b) depicts the synthesized oWFN modeling two airline instances.[5] This service receives two price requests from the agency addressed to the different instances (input places price.1 and price.2) and sends two price quotes. Then,

[5] This structure of this oWFN was slightly adjusted to simplify the presentation.

it waits to receive one ticket order (either on input place order.1 or order.2) and answers it accordingly. The resulting choreography would be deadlock-free. However, the airline's instances are not independent of each other. They are implicitly synchronized by the incoming arcs of the transitions receiving the orders. If this service had to be split into two services, explicit synchronization messages would have to be added to maintain deadlock-freedom. Still, the synthesized airline model can be seen as a starting point for further refinement.

Another issue of the partner synthesis are *causalities*. As sketched in the description of the generated traveler service (cf. Fig. 7(a)), a generated partner might send and receive messages in different — mostly less constraint — orders. This might yield to synthesized services which send acknowledgment messages before actually receiving the corresponding request. In such cases, the causality between the request and the acknowledgment is ignored. In [15], we introduced behavioral constraints into the synthesis process to rule out such implausible behavior.

Each of the participating services of both choreographies are controllable. As the first choreography shows, their composition may still deadlock. Such deadlocking scenarios are not obvious even for small choreographies. Therefore, design and verification of deadlock-free choreographies with a larger number of participants and/or more complex participant services are even more challenging if not impossible to do manually.

5 Case Study

In the previous sections, we analyzed the first and the second choreography (cf. Fig. 3 and Fig. 6, resp.) with five airline instances. For these five airlines, the resulting models already had over 3,000 states. The states space grows dramatically when the number of airlines is further increased (cf. Table 5). For ten airlines, the model has over nine million states, and for larger numbers, the full state space could not be constructed due to memory overflow[6] (denoted as '—' in Table 1).

However, several state space reduction rules can be applied to reduce the size of the state space while still being able to analyze desired properties such as deadlock-freedom. In our particular example, we applied *symmetry reduction* and the *partial order reduction*, both implemented in LoLA (see [12] for further references). The symmetry reduction exploits the fact that all airline instances have the same structure. This regular structure induces symmetries on the net structure itself, but also on the state space of the choreography. Intuitively, the instances act of the airline service act 'similar' or 'symmetric'. During the state space construction, symmetric states are merged. The partial order reduction follows a different approach: as all instances run concurrently, any order of transitions of the airline instances are represented in the state space. These

[6] The experiments where made with two gigabytes of memory.

Table 1. Net sizes (structurally reduced) and state spaces (full, reduced using symmetry reduction, reduced using partial order reduction, and reduced combining partial order reduction and symmetry reduction).

choreography	first example, cf. Fig. 3					second example, cf. Fig. 6				
airlines	1	5	10	100	1,000	1	5	10	100	1,000
places	20	63	113	1,013	10,013	19	63	113	1,013	10,113
transitions	10	41	76	706	7.006	12	52	97	907	9,007
states (full)	14	3,483	9,806,583	—	—	13	3,812	9,805,560	—	—
states (symmetry)	14	561	378,096	—	—	13	704	329,996	—	—
states (POR)	11	86	261	18,061	1,752,867	12	88	228	8,361	734,049
states (POR+symm.)	11	30	50	410	4,010	12	28	43	314	3,014

transition sequences introduce a lot of intermediate states. This is known as *state space explosion*. However, the actual order of independent actions is not relevant to detect deadlocks, for instance. To this end, partial order reduction tries to only construct one transition sequence (i. e., one order) of transitions of different airline instances to ease the state space explosion.

When each of the reduction techniques is applied in isolation, the state spaces grow more slowly, yet still exponentially in the number of airline instances. The combination of both techniques, however, yields a linear increase of states (cf. Table 5). Hence, we are able to verify properties of BPEL choreographies with thousands of participating services. This shows that the presented approach can be used to analyze real-life examples.

6 Conclusion

In this paper, we presented an analysis of choreographies expressed in BPEL4Chor based on Petri nets. Models of choreographies with a lot of participating services contain a lot of concurrency which results in state space explosion. Our experiments showed that the combination of several reduction techniques allows to handle choreographies with thousand participants.

Deadlocks in choreographies can be very subtle. In the introductory example, each participant was correct (i. e., controllable) by itself, but the composition introduced deadlocks. We showed how our tool chain helps to detect deadlocks in a reasonable time and thus ensures that the choreography can be executed.

Since a choreography is a closed world, the analysis techniques allow a participant to be generated out of other participants, which speeds up the choreography design. If an airline and a travel agency agree on their behavior, the customer has to comply with it and can neither force the airline nor the travel agency to adapt their behavior to his wishes.

All things considered, the analysis and synthesis approach are independent of BPEL as input language as the approaches are based on the formal model of

Petri nets. Therefore, the presented tool-chain (cf. Fig. 1) can be easily adapted to other service description languages.

6.1 Related Work

For analyzing BPEL4Chor choreographies, [16] presents a first approach for mapping BPEL4Chor to π-calculus. However, there was no formal mapping provided and it has not been shown whether the resulting π-formula can be verified in a reasonable period of time.

Choreographies themselves can be expressed by specifying (i) interconnection models and (ii) interaction models. An *interconnection model* captures the observable behavior of each participant in a choreography; that is, it defines an orchestration of the activities local to each participant. Activities of different participants are related in a choreography via message links tying together the local behavior into a global behavior. The basic messaging constructs are sending and receiving activities. BPMN [6] and BPEL4Chor are languages to express choreographies by interconnection models. An *interaction model* defines an ordering of the interactions of the processes on a global view. The basic messaging construct is the interaction activity, which models a message exchange between two participants. Current languages providing interaction models are Let's Dance [17] and WS-CDL [18]. Verification techniques are available for Let's Dance (cf. [19]) and WS-CDL (e. g. [20,21]). Since Let's Dance and WS-CDL provide interaction models, whereas BPEL4Chor provides interconnection models, the techniques cannot be directly applied to BPEL4Chor. [22] provides a formalization and a verification of BPMN models. However, BPMN does not originally support multiple instances of a participant as it is the case in BPEL4Chor.

[23] presents how to synthesize a BPEL processes which properly interacts with *one* given BPEL process. In contrast, we presented how to synthesize an oWFN out of n given BPEL processes.

6.2 Future Work

We plan to enhance and generalize the translation approach of [14] to synthesize a participant behavior description in BPEL instead of the oWFN only. For example, information about the participant topology has to be incorporated into the translation process to refine the resulting BPEL process.

Errors in choreographies can usually not be collated to a single participant, but to the combination of several participants. To this end, the repair of a erroneous choreography is nonlocal. We therefore plan to visualize the faulty scenario in the BPEL code of the affected participant(s) to support the designer in eliminating the detected problem.

In [9], the notion of *distributed controllability* was introduced. Distributed controllability focuses on synthesizing a set of services that interact deadlock-freely with a given service, and thus may allow to synthesize several independent instances of a participating service. This would ease the design of choreographies,

because as soon as the first participant and a participant topology is specified, the blueprints of the remaining participants can be synthesized. We plan to further investigate the first theoretical results whether they can be integrated into our approach.

Acknowledgments

The authors wish to thank Gero Decker for the discussions we had on the subject. Niels Lohmann and Oliver Kopp are funded by the German Federal Ministry of Education and Research (project Tools4BPEL, project number 01ISE08).

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, Organization for the Advancement of Structured Information Standards (OASIS) (April 2007)
2. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, IEEE Computer Society (July 2007)
3. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A.P., eds.: BPM 2006. Volume 4102 of Lecture Notes in Computer Science., Springer-Verlag (September 2006) 17–32
4. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. Data Knowl. Eng. (2007)
5. Chinnici, R., Gudgin, M., Moreau, J.J., Weerawarana, S.: Web services description language (WSDL) version 1.2 part 1: Core language. World Wide Web Consortium, Working Draft WD-wsdl12-20030611 (June 2003)
6. OMG: Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, Object Management Group (February 2006) `http://www.bpmn.org`.
7. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
8. Aalst, W.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
9. Schmidt, K.: Controllability of open workflow nets. In Desel, J., Frank, U., eds.: EMISA 2005. Volume 75 of Lecture Notes in Informatics (LNI)., Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), GI (2005) 236–249
10. Breugel, F.v., Koshkina, M.: Models and verification of BPEL. `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf` (September 2006)
11. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Techn. Report 212, Humboldt-Universität zu Berlin, Berlin, Germany (June 2007)
12. Schmidt, K.: LoLA: A low level analyser. In Nielsen, M., Simpson, D., eds.: ICATPN 2000. Number 1825 in Lecture Notes in Computer Science, Springer-Verlag (June 2000) 465–474
13. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Trans. Inf. and Syst. **E80-D**(3) (March 1997) 315–325

14. Lassen, K.B., Aalst, W.v.d.: WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL. In Meersman, R., Tari, Z., eds.: CooPIS 2006. Volume 4275 of Lecture Notes in Computer Science., Springer-Verlag (2006) 127–144
15. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. Lecture Notes in Computer Science, Springer-Verlag (September 2007)
16. Decker, G., Kopp, O., Puhlmann, F.: Service referrals in BPEL-based choreographies. In: 2nd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2007), University of Leicester (June 2007) 25–30
17. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: Let's dance: A language for service behavior modeling. In Meersman, R., Tari, Z., eds.: CooPIS 2006. Volume 4275 of Lecture Notes in Computer Science., Springer-Verlag (November 2006) 145–162
18. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation, W3C (November 2005) `http://www.w3.org/TR/ws-cdl-10`.
19. Decker, G., Zaha, J.M., Dumas, M.: Execution semantics for service choreographies. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: WS-FM 2006. Volume 4184 of Lecture Notes in Computer Science., Springer-Verlag (2006) 163–177
20. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration conformance for system design. In Ciancarini, P., Wiklicky, H., eds.: COORDINATION 2006. Volume 4038 of Lecture Notes in Computer Science., Springer-Verlag (2006) 63–81
21. Flavio Corredini, Francesco De Angelis, A.P.: Verification of WS-CDL choreographies. In: 2nd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2007), University of Leicester (2007) 13–18
22. Puhlmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In Dustdar, S., Fiadeiro, J.L., Sheth, A.P., eds.: BPM 2006. Volume 4102 of Lecture Notes in Computer Science., Springer-Verlag (2006) 145–160
23. Moser, S., Martens, A., Häbich, M., Mülle, J.: A hybrid approach for generating compatible WS-BPEL partner processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A.P., eds.: BPM 2006. Volume 4102 of Lecture Notes in Computer Science., Springer-Verlag (2006) 458–464