Local Enforceability in Interaction Petri Nets

Gero Decker, Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
(gero.decker,mathias.weske)@hpi.uni-potsdam.de

Abstract. In scenarios where a set of independent business partners engage in complex conversations, global interaction models are a means to specify the allowed interaction behavior from a global perspective. In these models atomic interactions serve as basic building blocks and behavioral dependencies are defined between them. Global interaction models might not be *locally enforceable*, i.e. they specify constraints that cannot be enforced during execution without additional synchronization interactions. As this property has only been defined textually so far, this paper presents a formal definition. For doing so, this paper introduces interaction Petri nets, a Petri net extension for representing global interaction models. Algorithms for deriving the behavioral interface for each partner and for enforceability checking are provided.

1 Introduction

In business-to-business collaboration scenarios different partners interact with each other in order to reach a common goal. Different means of interaction are possible such as exchanging faxes or phone calls. However, as more and more interaction is carried out through electronic messages, e.g. through web service invocations, there is a high need for establishing precise interaction specifications so that interoperability between the different partners' systems is guaranteed.

Choreographies are a means to describe interaction behavior from a global perspective. They enlist the allowed message exchanges as well as the control and data flow constraints between them. Once all partners have agreed on a choreography, the individual specifications for each partner (the behavioral interfaces) can be derived which in turn serve as starting point for adapting existing implementations and configurations or for implementing new services [7].

We can generally distinguish between two different styles for modeling choreographies. On the one hand we find *interaction modeling* languages where interactions are the basic building blocks and control and data flow dependencies are defined between them. In these models a certain atomicity of the message exchanges is assumed and only later on this atomicity is replaced by more detailed handling of asynchronism and exception handling. The Web Services Choreography Description Language (WS-CDL [11]) and Let's Dance [19] are examples for interaction modeling languages. On the other hand we find languages where message send and message receipt actions are the basic building blocks. Different actions are connected through control and data flow and corresponding send

and receive actions are connected through message flow. We call this second kind of choreography models *stitched behavioral interfaces*. The Business Process Modeling Notation (BPMN [1]) and basic Message Sequence Charts (MSC [10]) are examples for the second category. Also the Business Process Execution Language (BPEL [2]) falls into that category although it disqualifies as choreography language as it only shows the communication behavior of an individual process.

In contrast to stitched behavioral interfaces, where control flow is defined within the individual partners, certain anomalies are possible in choreography models when describing the control flow from a global point of view. Imagine a setting where partners C and D can only exchange a message after A has sent a message to B. Here, it is unclear how C knows when the message exchange between A and B has occurred. Only the introduction of additional synchronization messages could help to enforce the specified control flow. However, as a choreography should enlist all allowed interactions, it is unacceptable to leave the choreography unenforceable. All necessary interactions have to be factored into the model before it is used as specification for the collaboration.

Petri nets [17] are a popular formalism for describing the control flow aspects of processes. However, in the field of choreography modeling, Petri nets have only been used for modeling stitched behavioral interfaces. Therefore, this paper introduces interaction modeling using Petri nets.

The remainder of this paper is structured as follows. Section 2 introduces interaction Petri nets, before an algorithm for generating behavioral interfaces is introduced in section 3. Section 4 provides formal definitions for realizability and local enforceability of choreographies and section 5 presents algorithms for checking these properties. Section 6 reports on related work and section 7 concludes.

2 Interaction Petri Nets

This section introduces *interaction Petri nets*, a formal language to describe global interaction models based on classical place transition nets. Such nets consist of places and transitions that are connected through a flow relation. Places can contain tokens, which in turn are needed to enable transitions. Once a transition fires, tokens are consumed and produced. That way tokens flow through the net. In the Business Process Management world, Petri nets are used to describe business processes and workflows [17]. Here, transitions are interpreted as activities that are carried out by an actor. Workflow nets are a popular class of Petri nets, where there is exactly one input and one output place and every transition is on a path from the input to the output place.

In the case of interaction Petri nets, transitions are interpreted as interactions, i.e. message exchanges between two partners. These interactions are atomic in the sense that the send and receive actions are not decoupled but rather happen at the same time. An interaction Petri net specifies a set of valid

conversations, i.e. sequences of message exchanges. The following two subsections will present the formal model for interaction Petri nets and conversations.

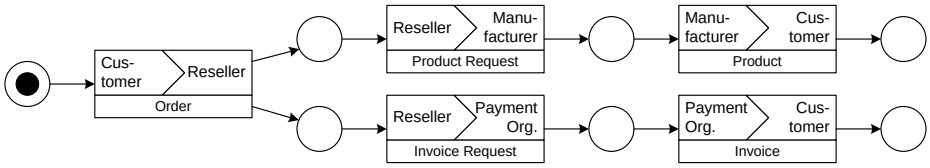

Fig. 1. Sample interaction Petri net

Figure 1 depicts a sample interaction Petri net. The visual representation of the interactions is inspired by the choreography language Let's Dance [19]: The upper left corner indicates the sending role, the upper right corner the receiving role and the bottom label the message type. In this example four partners engage in a conversation. A customer sends an order to a reseller, who then sends a product request to the manufacturer and an invoice request to the payment organization. Finally, the customer receives the product as well as the invoice.

2.1 Formal Model

In the remainder we will distinguish between *interaction models* and *interaction model occurrences*, where an interaction model is a triple of sender role, receiver role and message type. As several messages of the same type might be exchanged between the same sender and receiver in a conversation, we allow several occurrences of the same interaction model within an interaction Petri net.

Definition 1 (Interaction Petri net). *An interaction Petri net IPN is a tuple* $IPN = (P, T, F, R, s, r, MT, t, M_0)$ *where*

- P *is a set of places,*
- T *is a set of transitions (interaction model occurrences),*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation connecting places with interaction model occurrences,*
- R *is a set of roles,*
- $s, r : T \to R$ *are functions assigning the sender and the receiver role to an interaction model occurrence,*
- MT *is a set of message types,*
- $t : T \to MT$ *is a function assigning a message type to an interaction model occurrence and*
- $M_0 : P \to \mathbb{N}$ *is the initial marking.*

The set of interaction models IM for an interaction Petri net is defined as $IM := \{im \in R \times R \times MT \mid \exists t \in T\ (im = (s(t), r(t), t(t))\}$. We introduce the

auxiliary functions $roles : T \rightarrow \wp(R)$ where $roles(t) := \{s(t), r(t)\}$ and $in, out : T \rightarrow \wp(P)$ where $in(t) := \{p \in P \mid (p,t) \in F\}$ and $out(t) := \{p \in P \mid (t,p) \in F\}$.

We denote the marking of an interaction Petri net using functions $M : P \rightarrow \mathbb{N}$. If an interaction model occurrence t is enabled in marking M, i.e. it can fire, and actual firing results in marking M', we write $M \xrightarrow{t} M'$. A marking M' is said to be reachable from marking M if there is a (potentially empty) sequence of interaction model occurrences that lead from M to M', which we denote as $M \xrightarrow{*} M'$. We call a marking M a final marking, if there is no interaction model occurrence enabled in M. More details on execution semantics of Petri nets and reachability can be found in [17].

2.2 Conversations and Conformance

Choreographies can be interpreted in two ways:

1. **Interaction obligations.** Imagine an interaction Petri net with two interaction model occurrences arranged in a sequence where first an actor of role A sends a message $m1$ to B before B sends a message $m2$ back to A (depicted in Figure 2). When interpreting this choreography as interaction obligations, B must eventually send a message $m2$ back to A after having received a message $m1$ from A. A conversation that does not include a message exchange from B back to A would therefore not *conform* to the choreography.

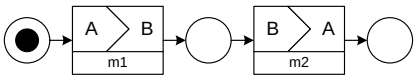

Fig. 2. Choreographies: obligations vs. constraints

2. **Interaction constraints.** When interpreting the interaction Petri net in Figure 2 as collection of interaction constraints, we would formulate that B can only send a message $m2$ back to A after it has received a message $m1$ from A. In this case a conversation only consisting of a message exchange from A to B would *weakly conform* to the choreography. Even an empty conversation would weakly conform to the choreography as none of the constraints is violated.

For defining conformance and weak conformance we introduce the following formal framework:

- I is a set of interaction instances,
- A is a set of actors,
- $s_I, r_I : I \rightarrow A$ are functions linking an interaction instance to the sending and receiving actors and
- $t_I : I \rightarrow MT$ is a function linking an interaction instance to a message type.

Definition 2 (Weak Conformance). *Let a conversation c be a sequence of interaction instances $i_1, \ldots, i_n$. Then c* weakly conforms *to an interaction Petri net IPN and a partial function $map : R \to A$ if and only if there exist markings $M_1, \ldots, M_n$ and interaction model occurrences $t_1, \ldots, t_n$ such that for all $j = 1, \ldots, n$:*

$$M_{j-1} \xrightarrow{t_j} M_j \ \wedge \ s_I(i_j) = map(s(t_j)) \wedge r_I(i_j) = map(r(t_j)) \wedge t_I(i_j) = t(t_j)$$

The fact that *map* is a partial function relating actors and roles implies that there is at most one actor playing a particular role in a conversation. However, one actor can also play different roles and there can be conversations where there is no actor playing a particular role. For the latter case imagine e.g. a procurement scenario where the seller sometimes carries out liability checks with a financial institute before actually delivering the goods. As the liability checks are optional, a financial institute does not necessarily appear in the conversation.

Definition 3 (Conformance). *A conversation c* conforms *to an interaction Petri net IPN and a partial function $map : R \to A$ if and only if c weakly conforms to (IPN, map) and marking M_n is a final marking.*

3 Generation of Behavioral Interfaces

An interaction Petri net can be used for modeling two distinct viewpoints in choreography design: (i) a choreography and (ii) a behavioral interface of a role r_i. In the latter case that particular role r_i must be involved in every interaction model occurrence, i.e. $\forall t \in T \ [r_i \in roles(t)]$.

1: **procedure** reduceIPN($IPN = (P, T, F, R, s, r, MT, t, M_0), r_i$)
2: **while** $t \in T \ (r_i \notin roles(t))$
3: **if** $\neg \exists t_2 \in T \ (in(t) \cap in(t_2) \neq \emptyset \wedge t_2 \neq t)$
4: **for each** $(p_1, p_2) \in in(t) \times out(t)$
5: $p_{new} := new(),\ P := P \cup \{p_{new}\}$
6: $M_0(p_{new}) := M_0(p_1) + M_0(p_2)$
7: $F := F \cup \{(t_2, p_{new}) \mid (p_1 \in out(t_2) \vee p_2 \in out(t_2)) \wedge t_2 \neq t\}$
8: $\cup \{(p_{new}, t_2) \mid p_2 \in in(t_2))\}$
9: $P := P \setminus (in(t) \cup out(t))$
10: **else**
11: **for each** $t_2 \in T \ (out(t) \cap in(t_2) \neq \emptyset \wedge t_2 \neq t)$
12: $t_{new} := new(),\ T := T \cup \{t_{new}\}$
13: $s(t_{new}) := s(t_2),\ r(t_{new}) := r(t_2),\ t(t_{new}) := t(t_2)$
14: $F := F \cup ((in(t) \cup (in(t_2) \setminus out(t))) \times \{t_{new}\})$
15: $\cup (\{t_{new}\} \times ((out(t) \setminus in(t_2)) \cup out(t_2))$
16: $T := T \setminus \{t\},\ F := F \cap (P \times T \cup T \times P)$
17: **return** $(P, T, F, R, s, r, MT, t, M_0)$

Fig. 3. Algorithm for generating the behavioral interface of role r_i

In a top-down choreography design process first the choreography is designed and agreed upon before behavioral interfaces are generated for all participants. A behavioral interface then serves as specification for the individual participant. If a participant already has a process implementation in place it must be checked whether it needs to be adapted to conform to the specification. If no process implementation is in place, the behavioral interface can serve as skeleton that needs to be refined. An overview of conformance relations between process specifications and implementations and typical refinements of specifications towards a complete implementation can be found in [6].

Figure 3 shows the algorithm for generating the behavioral interface of a role r_i out of a choreography. The basic idea is to identify all transitions (interaction model occurrences) where r_i is not involved and to then reduce the net accordingly. Simply marking these transitions as τ-transitions and leaving them in the net is not an option as choices might be possible between τ-transitions and other transitions. Imagine a choreography where role A can choose to either send a message to B or C where in the latter case C would then interact with B. From the perspective of B the interaction between A and C is not visible. Therefore, B only knows which path was taken as soon as a message of either A or C arrives. Figure 4 depicts this scenario.

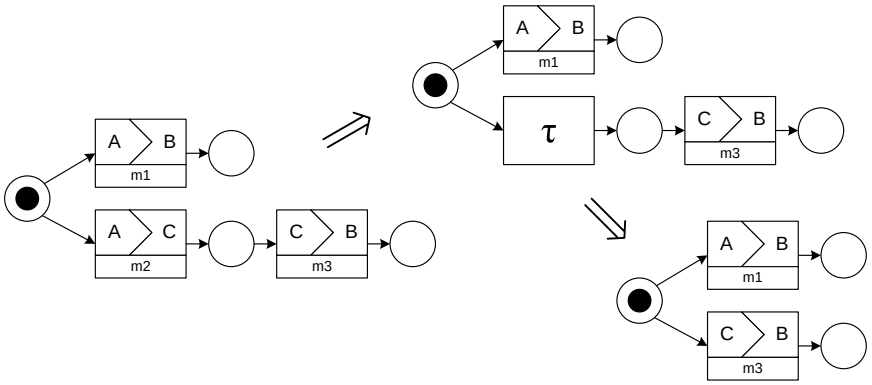

Fig. 4. τ-transitions need to be removed

We introduce two reduction rules. (i) The first can be found in lines 4-9 of Figure 3 and applies to those transitions t that do not share input places with other transitions. New places and flow relationships are introduced for connecting the preceding transitions with the succeeding transitions. Afterward, t and all places connected to it are removed (line 16). Figure 5 illustrates this rule.

(ii) The second rule covers those cases where t shares input places with other transitions (lines 11-15). Here, either one of these alternative transitions or one of the transitions succeeding t execute. If there are transitions t_2 succeeding t in parallel, then sequentialization has to be applied to the transitions t_2. This is

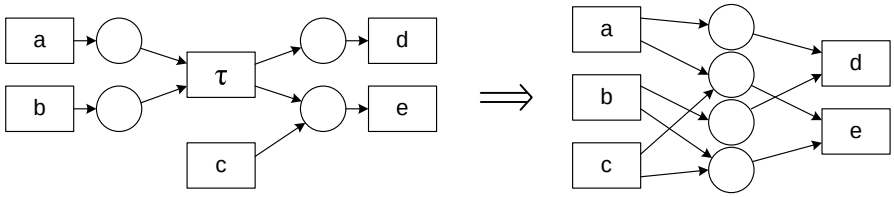

Fig. 5. Reduction rule (i)

achieved through the duplication of these transitions as illustrated in Figure 6. In contrast to the first rule no new places are created and none need to be removed. However, if there is no parallelism following t, the algorithm still duplicates transitions. This often leads to nets, where the transitions originally succeeding t are not reachable any longer. These transitions can easily be detected and removed for readability. Further optimization of the resulting nets would include the removal of redundant places and transitions.

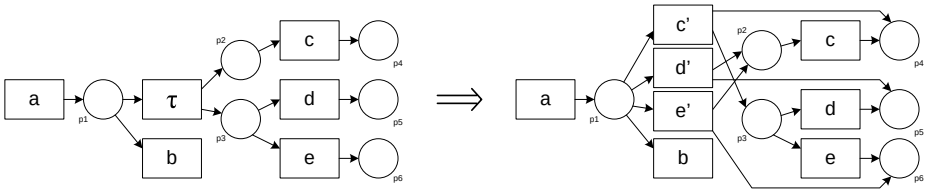

Fig. 6. Reduction rule (ii)

Actually, rule (i) could be skipped and rule (ii) could be applied to all cases. However, as rule (ii) flattens parallelism through the duplication of transitions, applying rule (i) where possible typically results in smaller nets.

The behavioral interface generated by the algorithm for role r_i exactly captures the choreography as seen by r_i. Therefore, it realizes a projection of a choreography *IPN* for role r_i. We denote this as $\pi_{r_i}(IPN)$.

4 Properties of Choreographies

This section will formally define the two properties *realizability* and *local enforceability* for interaction Petri nets. Both properties are essential for choreographies with more than two roles.

4.1 Realizability

Realizability is a well-known property in the space of software engineering. It is investigated whether a choreography can be implemented in a set of interacting processes in such a way that their composition exactly shows the specified

message exchange behavior [8]. The choreography is given in the form of a finite state machine in the referenced paper.

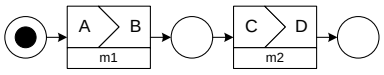

Fig. 7. Unrealizable sequence

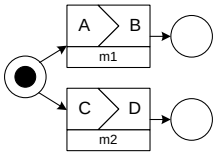

Fig. 8. Unrealizable choice

The realizability criterion can also be applied to interaction Petri nets. Figures 7 and 8 show two examples of unrealizable choreographies. In Figure 7 we see the problem that C and D cannot know if a message exchange between A and B has already occurred or not. When deriving the behavioral interfaces for the four roles using our algorithm from section 3, we would loose the control flow relationship between the two transitions. In Figure 8 we find a similar problem: This time the two message exchanges exclude each other. A and B cannot know whether the other message exchange has already happened between C and D and vice versa.

When bringing realizability to interaction Petri nets, we need to introduce the auxiliary function *JOIN* that composes a set of behavioral interfaces $bi_1, \ldots, bi_n$, where $bi_i = (P_i, T_i, F_i, R_i, s_i, r_i, MT_i, t_i, M_{0i})$, into a choreography *chor*. The definition is shown in Figure 9. We assume $\bigcap_{i=1}^{n} P_i = \emptyset$.

1: **procedure JOIN** $(bi_1, \ldots, bi_n)$
2: $P := \bigcup_{i=1}^{n} P_i$
3: $M_0 := \bigcup_{i=1}^{n} M_{0i}$
4: $R := \bigcup_{i=1}^{n} R_i$
5: $MT := \bigcup_{i=1}^{n} MT_i$
6: **for each** $u, v : \exists i, j \in 1, \ldots, n \ (i < j \wedge u \in T_i \wedge v \in T_j$
7: $\quad \wedge (s_i(u), r_i(u), t_i(u)) = (s_j(v), r_j(v), t_j(v))$
8: $\quad t_{new} := new(), T := T \cup \{t\}$
9: $\quad s(t_{new}) := s_i(u), r(t_{new}) := r_i(u), t(t_{new}) := t_i(u)$
10: $\quad F := F \cup ((in(u) \cup in(v)) \times \{t_{new}\}) \cup (\{t_{new}\} \times (out(u) \cup out(v)))$
11: **return** $(P, T, F, R, s, r, MT, t, M_0)$

Fig. 9. *JOIN* function for composing behavioral interfaces

Definition 4 (Realizability). *An interaction Petri net Chor is* realizable *if and only if there exists a set of behavioral interfaces $bi_1, \ldots, bi_n$ such that their composition $JOIN(bi_1, \ldots, bi_n)$ is branching bi-simulation related to Chor.*

Branching bi-simulation was introduced by van Glabbeek and Weijland in [18] and defines a semantic equivalence relation on process models. It respects

the branching structure of the models. Van der Aalst and Basten have shown in [14] how branching bi-simulation can be applied to labeled Petri nets. Interaction Petri nets can be seen as special kind of labeled Petri nets where the label is composed of sender role, receiver role and message type.

4.2 Local Enforceability

The example in Figure 10 is not realizable, either. However, this time we can create corresponding behavioral interfaces the composition of which only produces traces (conversations) that conform to the choreography. Figure 11 illustrates that the interactions involving B / C and C / D are arranged in a sequence in the behavioral interface bi_C.

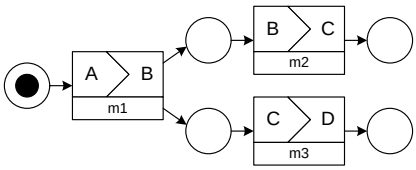

Fig. 10. Unrealizable but locally enforceable choreography

We see that realizability is very restrictive as it demands that *all* conversations produced by the choreography must also be produced by the composition of the behavioral interfaces. Therefore, realizability would not allow an additional control flow dependency as it can be found in bi_C.

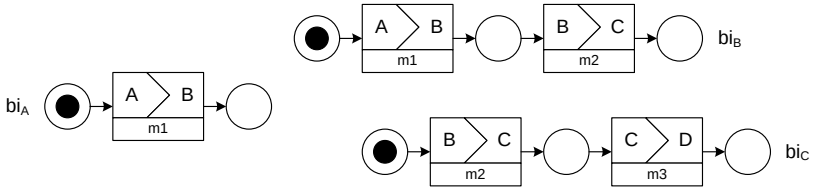

Fig. 11. Behavioral interfaces introducing an additional constraint

Choreographies are typically treated as collection of obligations in the sense that termination of conversations as specified in the choreography is required. We resort to the notion of local enforceability as initially presented in [20] for checking choreographies. There, a locally enforceable choreography is textually defined as follows: "The global model can be mapped into local ones in such a way that the resulting local models satisfy the following two conditions: (i) they contain only interactions described in the global model; and (ii) they are able to collectively enforce all the constraints expressed in the global model."

In the remainder of this section we are going to introduce a formal definition for local enforceability. As the textual definition leaves some ambiguities, e.g. concerning the question whether all or at least some interactions in the choreography must be reachable in the behavioral interfaces and whether local models are required that actually reach a conversation end as specified in the choreography. In order to eliminate these ambiguities we also take the enforceability checking algorithm into account that was also introduced in [20].

As it is demanded that all constraints are enforced in the global model, we disallow message exchanges in a marking of the composition of the behavioral interfaces that are not allowed in the corresponding markings of the choreography. This can be checked using simulation techniques as presented in [18]. The choreography must simulate all behavior possible in the composition of the behavioral interfaces. This enables the introduction of additional constraints as shown in Figure 11. However, a drawback of simulation techniques is that proper termination is not checked. A choreography consisting of a sequence of four interaction model occurrences would of course be simulation-related to a composition of behavioral interfaces where only the first two interaction model occurrences appear. This is not desired and we therefore introduce the additional condition that a final marking has to be reached in the choreography every time a final marking is reached in the composition of the behavioral interfaces. We call such an extended simulation relation *termination similarity*.

Definition 5 (Local Enforceability). *An interaction Petri net* $Chor = (P_C, T_C, F_C, R_C, s_C, r_C, MT_C, t_C, M_{0C})$ *where all interaction model occurrences are reachable is* locally enforceable *if and only if there exists a set of behavioral interfaces* $bi_1, \ldots, bi_n$ *such that their composition* $BI = JOIN(bi_1, \ldots, bi_n) = (P_B, T_B, F_B, R_B, s_B, r_B, MT_B, t_B, M_{0B})$ *fulfills the following three criteria:*

1. *the set of interaction models is equal for* $Chor$ *and* BI*, i.e.* $IM_C = IM_B$*,*
2. *for every interaction model there is at least one reachable interaction model occurrence in* BI*, i.e.* $\forall im \in IM_C\ [\exists u \in T_B, M, M'\ (im = (s_B(u), r_B(u), t_B(u)) \wedge M_{0B} \xrightarrow{*} M \wedge M \xrightarrow{u} M')]$*, and*
3. $Chor$ termination simulates BI*, i.e.* $BI \leq_{ts} Chor$*.*

Definition 6 (Termination Similarity). *Two interaction Petri nets* $IPN_1 = (P_1, T_1, F_1, R_1, s_1, r_1, MT_1, t_1, M_{01})$ *and* $IPN_2 = (P_2, T_2, F_2, R_2, s_2, r_2, MT_2, t_2, M_{02})$ *are* termination similar*, denoted* $IPN_1 \leq_{ts} IPN_2$*, if and only if there exists a relation* R *between the markings of* IPN_1 *and* IPN_2 *such that:*

1. $M_{01}\ R\ M_{02}$*,*
2. $M_1\ R\ M_2 \wedge M_1 \xrightarrow{u} M_1' \Rightarrow \exists v \in T_2, M_2'\ (M_2 \xrightarrow{v} M_2' \wedge M_1'\ R\ M_2' \wedge (s_1(u), r_1(u), t_1(u)) = (s_2(v), r_2(v), t_2(v)))$ *and*
3. $M_1\ R\ M_2 \wedge \neg\exists u \in T_1\ (M_1 \xrightarrow{u} M_1') \Rightarrow \neg\exists v \in T_2, M_2'\ (M_2 \xrightarrow{v} M_2')$*.*

It is obvious that all realizable choreographies are also locally enforceable, i.e. the set of realizable choreographies is a subset of the set of locally enforceable choreographies.

5 Detection Algorithms

This section will show how choreographies can be checked for the properties introduced in the previous section.

Detection of Realizability. The algorithm for generating behavioral interfaces from section 3 can be used for checking realizability. For each role involved in a choreography the corresponding behavioral interface is generated. These behavioral interfaces are composed using the *JOIN* function. The result is compared with the original choreography using branching bi-simulation.

$$JOIN(\pi_{r_1}(IPN), \ldots, \pi_{r_n}(IPN)) \sim_b IPN$$

Detection of Local Enforceability. Figure 12 presents the algorithm for checking local enforceability. It is restricted to interaction Petri nets that are bounded, i.e. for every place there is a maximum number of tokens. This restriction leads to the fact that the number of reachable markings is finite. The algorithm is recursively defined and runs through all reachable markings of the interaction Petri net. The basic idea is to identify the two different sources of unenforceability always caused by pairs of transitions u, v not sharing a common role: (i) u disables v or (ii) u enables v. The first case cannot be solved by adding synchronization interactions or adding further control flow dependencies. Lines 9 and 10 detect this case.

The second case, where a transition u enables v and u and v do not share a common role, can be resolved in the following manner: The execution of v is always delayed until another transition w fires that shares a common role with v. This delay is realized through *blocked* in the algorithm. If a transition is blocked in a marking M we do not investigate the case that it fires. If a transition v that is not enabled in marking M becomes enabled in marking M' and $M \xrightarrow{u} M'$ then it is added to the set *blocked* (line 12). A transition can be unblocked as soon as a transition fires that shares a role with v (line 11).

Sometimes there is no chance of reaching the end of a conversation without unblocking a transition. This indicates that an unenforceable control flow dependency is present. *wasReached* indicates which interaction models have already been reached while traversing the reachable markings. In order to support cyclic Petri nets, we have to keep track of which markings have already been visited. This is realized through *visited*. $visited_{cancel}$ contains all those markings where transitions are enabled but all of them are either blocked or lead to other markings in $visited_{cancel}$. As different paths might lead to the same marking but with different blocking history, we allow to visit a marking several times – once per blocking configuration. As the number of blocking configurations and the number of markings are finite we can conclude that the algorithm always terminates. In the worst case there are $2^{|T|}|\mathcal{M}|$ combinations to be checked, where $\mathcal{M}$ is the set of all reachable markings. However, for most choreography examples we have verified, the computational complexity of the algorithm is close to linear to $|\mathcal{M}|$.

```
 1: procedure checkEnforceability (IPN)
 2:
 3:     procedure recursivelyCheck (M, blocked)
 4:     if (M, blocked) ∈ visited
 5:         return true
 6:     visited := visited ∪ {(M, blocked)}
 7:     completed := false
 8:     for each u ∈ T, M' : M -u-> M' ∧ u ∉ blocked
 9:         if ∃v ∈ enabled(M) \ enabled(M') (roles(u) ∩ roles(v) = ∅)
10:             return false
11:         blocked' := (blocked \ {v ∈ T | roles(u) ∩ roles(v) ≠ ∅}) ∪
12:             {v ∈ enabled(M') \ enabled(M) | roles(u) ∩ roles(v) = ∅}
13:         if ¬ recursivelyCheck (M', blocked')
14:             return false
15:         if (M', blocked') ∉ visited_cancel
16:             wasReached := wasReached ∪ {(s(u), r(u), t(u))}
17:             completed := true
18:     if ¬completed ∧ enabled(M) ≠ ∅
19:         visited_cancel := visited_cancel ∪ {(M, blocked)}
20:     return true
21:
22: visited := ∅, visited_cancel := ∅, wasReached := ∅
23: return ( recursivelyCheck (M_0, ∅) ∧ wasReached = IM)
```

Fig. 12. Algorithm for detecting locally unenforceable choreographies

The auxiliary function *enabled* is used throughout the algorithm, linking a set of transitions to a marking M in the following way: $enabled(M) = \{u \in T \mid \exists M' \ (M \xrightarrow{u} M')\}$.

6 Related Work

Endpoint-oriented formalisms for describing interacting systems are more common than formalisms following the interaction-centric paradigm. In the field of software engineering it is very common to describe interacting components using communicating finite state machines (FSM). However, problems of state explosion in the case of bounded message queues and undecidability for unbounded message queues has motivated the use of conversation models. Here, the control flow between the interactions is seen from the global perspective in the sense that all send events are related in one FSM. It has been reported that for an important subclass of communicating FSMs there is a conversation model where the described interaction behavior is equivalent to behavior of the interconnected FSMs. This synchronizability is introduced in [9]. Realizability of conversation models is studied in [8].

Most work on choreography modeling using Petri nets uses the stitched behavioral interfaces approach, e.g. [16, 7]. Behavioral interfaces can be represented

using workflow modules, a subclass of Petri nets where a distinction is made between internal, input and output places [12]. Input and output places represent inbound and outbound message queues for communication with the environment. Different workflow modules are stitched together for carrying out compatibility checking.

π-calculus is a popular endpoint-centric process algebra especially suited for describing interacting processes [13]. There has been work on process algebras for interaction modeling. The work presented in [5] was driven by the Web Service Choreography Description Language (WS-CDL [11]). Another global calculus is presented in [4].

The different interpretations of choreographies regarding obligations and constraints are introduced in [3]. DecSerFlow [15] is a graphical notation for describing constraints in processes. The constructs introduced in the language translate into linear temporal logic expressions. Let's Dance [19] is a graphical choreography language that incorporates both notions of obligations and constraints. Constructs like for each repetitions rather express obligations, whereas also constructs for inhibition, a useful means for expressing constraints, are also part of the language. The property of local enforceability was only reported for Let's Dance so far [20]. However, as mentioned above, no formal definition is available for this property.

7 Conclusion

This paper has presented interaction Petri nets, an extension for interaction modeling based on Petri nets along with an algorithm for deriving corresponding behavioral interfaces. Furthermore, the two properties realizability and local enforceability have been defined for interaction Petri nets and algorithms for checking choreographies have been introduced. We have validated the proposed algorithms through implementation. Our GMF[1]-based tool is a graphical editor for interaction Petri nets. Generated behavioral interfaces can be neatly visualized thanks to automatic layouting functionality. Figure 13 shows a screenshot of the editor.

Throughout this paper, message exchanges were assumed to be atomic, i.e. we assumed synchronous communication. As a next step we will consider the impact of asynchronism on choreographies. What effect does it have if the derived behavioral interfaces are distributed to the different partners and executed with asynchronous communication?

In future work we are going to investigate mappings from higher level interaction modeling languages such as WS-CDL and Let's Dance to interaction Petri nets.

Acknowledgments. We would like to thank Marlon Dumas for his valuable input on this topic.

[1] See http://www.eclipse.org/gmf/

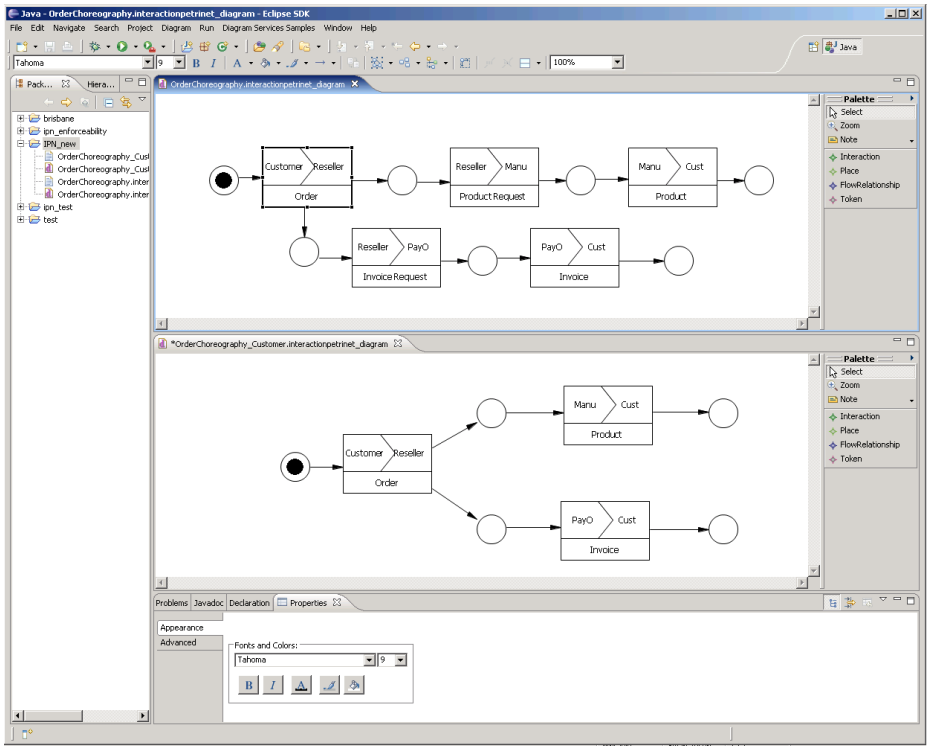

Fig. 13. Screenshot of the graphical editor for interaction Petri nets

References

1. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. http://www.bpmn.org/.
2. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, version 1.1. Technical report, OASIS, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.
3. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. A priori conformance verification for guaranteeing interoperability in open environments. In *Proceedings of 4th International Conference on Service Oriented Computing, Chicago, USA (4th–7th December 2006)*, volume 4294 of *LNCS*, pages 339–351, 2006.
4. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and Orchestration: A Synergic Approach for System Design. In *Proceedings 3rd International Conference on Service Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands, Dec 2005. Springer Verlag.
5. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *Proceedings 16th European Symposium on Program-*

ming (ESOP) as part of the European Joint Conferences on Software Theory and Practice (ETAPS), Braga, Portugal, March 2007.
6. G. Decker and M. Weske. Behavioral Consistency for B2B Process Integration. In *Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, LNCS, Trondheim, Norway, June 2007. Springer Verlag.
7. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
8. X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and analysis of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, November 2004.
9. X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Trans. Softw. Eng.*, 31(12):1042–1055, 2005.
10. ITU-T. Message sequence chart. Recommendation Z.120, ITU-T, 2000.
11. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report, November 2005. http://www.w3.org/TR/ws-cdl-10.
12. A. Martens. Analyzing Web Service based Business Processes. In M. Cerioli, editor, *Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05), Part of the 2005 European Joint Conferences on Theory and Practice of Software (ETAPS'05)*, volume 3442 of *Lecture Notes in Computer Science*, Edinburgh, Scotland, April 2005. Springer-Verlag.
13. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, 100:1–40, 1992.
14. W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, uary.
15. W. M. P. van der Aalst and M. Pesic. Decserflow: Towards a truly declarative service flow language. In *Proceedings 3rd International Workshop on Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *LNCS*, pages 1–23, Vienna, Austria, September 2006. Springer.
16. W. M. P. van der Aalst and M. Weske. The P2P Approach to Interorganizational Workflows. In *Proceedings of the 13th Conference on Advanced Information Systems Engineering (CAiSE'01)*, number 2068 in LNCS, pages 140–156. Springer Verlag, 2001.
17. W. v. d. van der Aalst and K. v. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, January 2002.
18. R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.
19. J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. A Language for Service Behavior Modeling. In *Proceedings 14th International Conference on Cooperative Information Systems (CoopIS 2006)*, Montpellier, France, Nov 2006. Springer Verlag.
20. J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker. Service Interaction Modeling: Bridging Global and Local Views. In *Proceedings 10th IEEE International EDOC Conference (EDOC 2006)*, Hong Kong, Oct 2006.