Service Interaction Modeling: Bridging Global and Local Views

Johannes Maria Zaha[*], Marlon Dumas[†], Arthur ter Hofstede
Queensland University of Technology, Brisbane, Australia
(j.zaha,m.dumas,a.terhofstede)@qut.edu.au

Alistair Barros
SAP Research Centre, Brisbane, Australia
alistair.barros@sap.com

Gero Decker[‡]
Hasso-Plattner-Institute, Potsdam, Germany
gero.decker@hpi.uni-potsdam.de

Abstract

In a Service-Oriented Architecture (SOA), a system is viewed as a collection of independent units (services) that interact with one another through message exchanges. Established languages such as the Web Services Description Language and the Business Process Execution Language allow developers to capture the interactions in which an individual service can engage, both from a structural and from a behavioral perspective. However, in large service-oriented systems, stakeholders may require a global picture of the way services interact with each other, rather than multiple small pictures focusing on individual services. Such "global models" are especially useful when a set of services interact in such a way that none of them sees all messages being exchanged, yet interactions taking place between some services affect the way other services interact. An issue that arises when dealing with global models of service interactions is that these models may capture behavioral constraints that can not be enforced locally. In other words, some global models may not be translatable into a collection of local models such that the sum of the local models equals the original global model. Starting from a previously proposed language for global modeling of service interactions, this paper defines an algorithm for determining if a global model is locally enforceable and an algorithm for generating local models from global ones.

1 Introduction

As the first generation of web service technology based on XML, SOAP, and WSDL reaches a certain level of maturity, a second generation targeting long-running collaborative business processes is gestating. In the first generation, web services are equated to sets of operations and message types (cf. WSDL). This conception reflects an emphasis on single request-response interactions. Meanwhile, the second generation of web service technology targets conversational interactions, with service descriptions capturing not only individual message exchanges and the underlying message types, but also dependencies between these exchanges, most notably behavioral dependencies.

This trend is evidenced by the emergence of languages for modeling and implementing services that can engage in conversational interactions. These languages include the Business Process Modeling Notation (BPMN) [15] and the Business Process Execution Language for Web Services (BPEL) [1], which respectively target the design and the implementation phases of service development. Other language definition initiatives in this space are the Web Services Choreography Description Language (WS-CDL) [10] and variants of UML Activity Diagrams such as BPSS/ebBP [6] and UML Profile for EDOC [12].

An analysis of approaches to conversational service modelling unveils two different approaches. On the one hand, interactions between conversational services can be modeled from the perspective of individual services, that is, each individual service model defines the set of messages that the service in question can send and receive; individual models are then "stitched together" to capture conversations between multiple services. This approach is suitable when the goal is to build individual services or to service-enable an existing application. However, during the early phases of the service development lifecycle, emphasis is not on building individual services but rather on identifying potential services and understanding and analyzing their interactions. In these phases, stakeholders need a global picture of the way services interact with each other. Thus, models at this level need to emphasize the interactions between services

[*] Funded in part by SAP
[†] Funded by a "Smart State" Fellowship co-sponsored by Queensland Government and SAP
[‡] Research conducted while with SAP Research Centre Brisbane

and their interdependencies. Such "global models" are especially useful when a set of parties interact in such a way that none of them sees all messages being exchanged, yet interactions taking place between some parties have an impact on the way other parties interact. WS-CDL is a typical example of a language for describing global models of service interactions (also known as *choreographies*).

Given their complementarity, an approach to conversational service modeling that seamlessly integrates global and local views, is desirable. In previous work [17], we introduced a language, namely "Let's Dance", for modeling service interactions and their behavioral dependencies. Let's Dance supports service interaction modeling both from a global and from a local viewpoint. In a global (or choreography) model, interactions are described from the viewpoint of an ideal observer who oversees all interactions between a set of services. On the other hand, local models focus on the perspective of a particular service, capturing only those interactions that directly involve it. A possible usage scenario is one where global models are produced by analysts to agree on interaction scenarios from a global perspective, while local models are produced during system design and handed on to implementers. Implementers then refine the local models and/or use them to generate code templates (e.g. in BPEL). To ensure proper handover between these users, it is necessary to have a mapping from global to local models and/or to be able to check that a local model is consistent with a global one.

It turns out that not all global models 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. For example, consider a global interaction model containing: (i) a first interaction where an actor A sends a message to an actor B; (ii) a second interaction where an actor C sends a message to an actor D; and (iii) a constraint to the effect that the second interaction can not occur before the first one. An obvious question arises then: How can actors C and/or D know that the interaction between A and B has taken place in the absence of any interaction between actors A and B on the one hand, and actors C and D on the other? Thus, either the model needs to be enhanced with an interaction between A/B and C/D, or the sequential execution constraint will not be enforced. In WS-CDL, constraints that can not be enforced using the explicitly declared interactions are enforced by implicit interactions. Since we envisage business analysts "signing off" on global models defined in Let's Dance, we consider the option of introducing such "hidden interactions" implausible. Hence, tools for global service modeling need to ensure that global models are "locally enforceable", meaning that they can be translated into sets of local models satisfying the two conditions above. This paper presents an algorithm for determining whether or not a global model expressed in Let's Dance is locally enforceable. The paper also provides an algorithm to translate global models into local ones.

The paper is structured as follows. Section 2 gives an overview of the Let's Dance language. Sections 3 and 4 respectively present the algorithms for determining the local enforceability of global models and for generating local models from global models. Section 5 discusses related work and Section 6 concludes.

2 Language Overview

Let's Dance abstracts away from implementation details and avoids reliance on imperative programming constructs. In particular, the language does not rely on variable assignment, while conditions need not be written in an executable language. Instead, these are treated as free-text labels that are subsequently refined. Still, models defined in Let's Dance can be used to generate BPEL templates or to check that a service implementation conforms to the behavioral constraints captured in a model. This section provides an overview of Let's Dance. A more detailed description of Let's Dance and solutions for the service interaction patterns presented in [2] can be found in [17].

2.1 Language Constructs

A choreography consists of a set of interrelated service interactions corresponding to message exchanges. At the lowest level of abstraction, an interaction is composed of a message sending action and a message receipt action (referred to as communication actions). Communication actions are represented by non-regular pentagons (symbol ▷ for send and ▷ for receive) that are juxtaposed to form a rectangle denoting an elementary interaction. As illustrated in Figure 1, a communication action is performed by an actor playing a role, specified at the top corner of a communication action. Roles are written in uppercase and the actor playing this role (the "actor reference") is written in lowercase between brackets. The name of the message type for the receive actions can be omitted (since the same type applies for both send and receive).

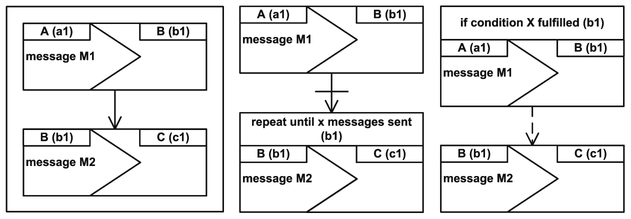

Figure 1. Constructs of Let's Dance

Interactions can be inter-related using the constructs depicted in Figure 1. The relationship on the left-hand side is called “precedes” and is depicted by a directed edge: the source interaction can only occur after the target interaction has occurred. That is, after the receipt of a message “M1” by “B”, “B” is able to send a message “M2” to “C”. The rectangle surrounding these two interactions denotes a composite interaction, which can be related with other interactions with any type of relationship. The relationship at the center of the figure is called “inhibits”, depicted by a crossed directed edge. It denotes that after the source interaction has occurred, the target interaction can no longer occur. That is, after “B” has received a message “M1” from “A”, it may not send a message “M2” to “C”. The latter interaction can be repeated until “x” messages have been sent, which is indicated by the header on top of the interaction. The actor executing the repetition instruction is noted in brackets. Finally, the relationship on the right-hand side of the figure, called “weak-precedes”, denotes that “B” is not able to send a message “M2” until “A” has sent a message “M1” or until this interaction has been inhibited. That is, the target interaction can only occur after the source interaction has reached a final status, which may be “completed” or “skipped” (i.e. “inhibited”). In the example, the upper interaction has a guard assigned, which is denoted by the header on top of the interaction. This interaction is only executed if the guard evaluates to true. The actor who evaluates the guard is noted in brackets.

2.2 Example

An example of a choreography corresponding to a loan application process is depicted in Figure 2. A client “c” sends a (loan) application to the loan department “l” of a financial institution. Once this interaction is completed, a composite interaction is enabled. This composite interaction contains two guarded sub-interactions. The two elementary interactions on the left-hand side take place only if a credit check is requested. If so, the loan department sends a message “check credit” to the Bureau of Credit Registration (BCR) and receives the credit information for the requested client. If no credit check is requested and the guard evaluates to false, both elementary interactions are inhibited. The two elementary interactions on the right-hand side are executed only if the loan department requests an optional insurance. Since a composite interaction is completed if all sub-interactions have been executed or inhibited, the succeeding interactions are enabled even if both guards inside the composite interaction evaluate to false. For connecting the two following interactions with the composite interaction, a connector has been used for multiple arrows. The loan department either sends a rejection of the application to the client or issues a request for payment with the payment department, whereby here a two-way inhibits relationship is used. If a request for payment is issued, payment notifications for each of the accounts nominated by the client are sent. This is captured through the repetition of the interaction using an (informally) specified condition in the box at the top corner of the interaction.

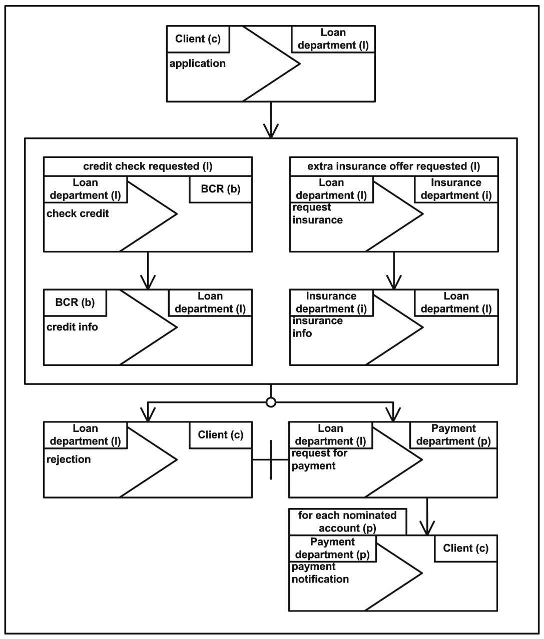

Figure 2. Choreography of a loan application

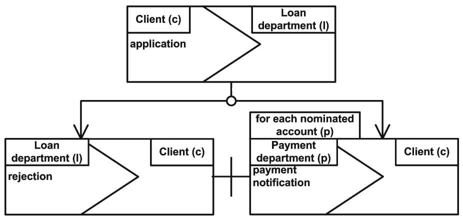

Figure 3. Local model for the client

Figure 3 shows the loan application choreography from the view of the actor “c”, playing the role of a client. This local model starts with an interaction for the client sending an application to the loan department, followed by the receipt of either a rejection or payment notifications. The two-way inhibits relationship is a derived relationship that does not explicitly appear in the global model (i.e. it is a “derived” relation). The same holds for the precedes relationship that connects to the interaction for sending the payment notifications, since the client is not involved in any interaction occurring between these two. This illustrates that local models may reflect relationships that are derived from, but not explicitly represented in the global model.

2.3 Abstract Syntax

The abstract syntax of the language is formally captured by the following definition of a Let's Dance choreography.

Definition 1. A *Choreography* (or *Global Model*) is a tuple (I, RI, GI, A, c_0, $Precedes$, $WeakPrecedes$, $Inhibits$, $Parent$, $Performs$) such that: *(a) I* is a set of Interactions; *(b) E* is a set of Expressions; *(c) RE* $\subseteq$ S is a set of Repeat-Expressions; *(d) GE* $\subseteq$ S is a set of Guard-Expressions; *(e) A* is a set of Actors; *(f)* $c_0 \in$ I is the top-level interaction of the choreography; *(g) Precedes*, *WeakPrecedes*, *Inhibits* $\subseteq$ I $\times$ I are three binary relations over the set of interactions I; *(h) Parent* $\subseteq$ I $\times$ I is the relation between interactions and their direct sub-interactions; *(i) Assignments* $\subseteq$ I $\times$ E is the relation between interactions and expressions; *(j) Performs*: I $\rightarrow \wp$(A) is a partial function linking interactions to actors; *(k) Conducts*: S $\rightarrow \wp$(A) is a partial function linking expressions to actors.

In the following definition, the symbol $Ancestor$ denotes the transitive closure of relation $Parent$, i.e. $Ancestor = Parent^{+}$. The sets *RI* $\subseteq$ I and *GI* $\subseteq$ I are used as abbreviations for Repeated Interactions and Guarded Interactions respectively and are defined as follows:

- $RI = \{r \in I \mid \exists e \in \; RL\,[(r, e) \in Assignments]\}$
- $GI = \{g \in I \mid \exists e \in \; GL\,[(g, e) \in Assignments]\}$

The definition of a well-formed choreography below captures certain syntactic constraints that exclude some incorrect choreographies. In the rest of the paper, we assume all choreographies to be well-formed.

Definition 2. A choreography $C = (I,\ RI,\ GI,\ A,\ c_0, Precedes,$*WeakPrecedes*$, Inhibits, Parent, Performs)$ is well-formed if:

- c_0 has no parent: $\neg\exists i \in I[i\ Parent\ c_0]$
- Each interaction other than the root has one and only one parent: $\forall i \in I \setminus \{c_0\}\ \exists! j \in I[j\ Parent\ i]$
- No relation that starts inside a repeated (composite) interaction crosses the boundary of this interaction: $\forall i, j \in I\ \forall k \in$ *RI*$[(k\ Ancestor\ i \wedge (i\ Precedes\ j \vee i$ *WeakPrecedes* $j \vee i\ Inhibits\ j))$ $\rightarrow k\ Ancestor\ j]$
- There are no "precedence dependencies" between ancestors and descendants: $Ancestor \cap (Precedes \cup$*WeakPrecedes*$) = \emptyset$
- There are no cyclic precedence dependencies: $Precedes \cup$*WeakPrecedes*$\cup Parent$ is acyclic
- An interaction involves at most two actors: $\forall i \in I[1 \leq |Performs(i)| \leq 2]$

3 Local Enforceability

As previously discussed, a choreography may include relationships that are not locally enforceable. Figure 4 shows a sample choreography with non-locally enforceable relationships.

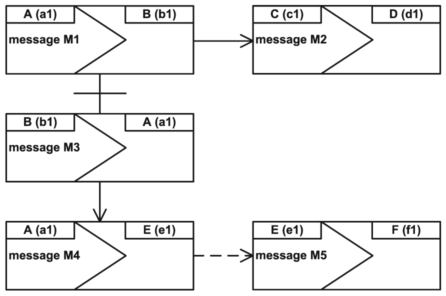

Figure 4. Non-enforceable choreography

The precedes relationship between the two elementary interactions on top of the figure is not enforceable. This relationship denotes that the sending of a message "M2" can only occur after the sending of a message "M1" (as perceived by an ideal global observer). Since every communication action is carried out by a different actor, it is not possible for actors "c1" or "d1" to know when has a message "M1" been sent. On the other hand, the two-way inhibits relationship between the interactions labeled "M1" and "M3" is enforceable, since there are two common actors performing these interactions, namely "a1" and "b1", and these actors can enforce the inhibits relationship in their respective local models. The same holds for the precedes relationship connecting the interactions labeled "M3" and "M4". These two interactions only share one common actor, namely "a1", but since "a1" will not start sending a message "M4" until it has received a message "M3", the precedes relationship can be enforced in the local model of "a1". The source and target interactions of the weak-precedes relationship at the bottom of the figure also share a common actor, namely "e1". Nevertheless this relationship is not enforceable. Indeed, according to the meaning of the weak-precedes relation, message "M5" may need to be exchanged even if the source interaction of the weak-precedes relationship has been skipped. Thus, "e1" needs to know when this source interaction is skipped. In the depicted choreography the source interaction will be skipped if the interaction labeled "M1" occurs, because in this case the interaction labeled "M3" and therewith also the interaction labeled "M4" will never occur due to the inhibits relationship. Since "e1" is not involved in the interaction labeled "M1", it will never know that the source interaction of the weak-precedes relationship has been skipped. The situation would be different if the interaction labeled "M2" was performed by "b1" and "e1". In this case "e1" would know that the interaction for sending a message "M4" has been skipped, as soon as it sees a message "M2", and thus "e1" can enforce the weak-precedes relationship.

The above example illustrates the main issues for checking enforceability. In this section, we present an algorithm for checking the local enforceability of relationships between elementary interactions. This algorithm relies on two auxiliary algorithms: (i) an algorithm for "expanding" a choreography into an equivalent choreography in which every relationship involves only elementary interactions (as opposed to composite interactions); and (ii) an algorithm for detecting interactions in a choreography that will never be executed due to conflicting relationships in the choreography. The definitions of these algorithms rely on the following notations:

- $EI \subseteq I$ is the set of *Elementary Interactions*
  $EI = \{i \in I \mid \neg\exists j[i\ Parent\ j]\}$
- $CI \subseteq I$ is the set of *Composite Interactions*
  $CI = I \setminus EI$
- $Parent(i)$ is the parent of interaction i
  $\forall i \in I \setminus \{c_0\}\ Parent(i) =$ the only element in the set $\{p \in I \mid p\ Parent\ i\}$
- $Ancestors(i)$ is the set of ancestors of i
  $Ancestors(i) = \{a \in I \mid a\ Parent^+\ i\}$
- $Initial : CI \rightarrow \wp(EI)$ computes the set of elementary interactions in a composite interaction that are not target of control dependencies from other interactions in the same composite interaction:
  $Initial(ci) = \{ei \in EI \mid ci\ Ancestor\ ei \wedge \neg\exists k, m \in I[ci\ Ancestor\ k \wedge (k, m) \in Precedes \cup WeakPrecedes \wedge (m\ Ancestor\ ei \vee m = ei)]\}$
- $Final : CI \rightarrow \wp(EI)$ computes the set of elementary interactions in a composite interaction that are not the source of control dependencies leading to other interactions in the same composite interaction:
  $Final(ci) = \{ei \in EI \mid ci\ Ancestor\ ei \wedge \neg\exists k, m \in I[(ci\ Ancestor\ k) \wedge (m, k) \in (Precedes \cup WeakPrecedes) \wedge (m\ Ancestor\ ei \vee m = ei)]\}$

3.1 Choreography expansion

The algorithm for expanding composite interactions is presented in Figure 5. This algorithm first adds every pair of interactions to the set of precedes relationships that has a composite interaction in between and where there exists a consecutive set of precedes relationships connecting these three interactions. This auxiliary construct is introduced in order to detect the enforceability of expanded interactions, introduced below. After that the relationships originating from a composite interaction are treated. Lines 2 to 7 of the algorithm denote the substitution of all relationships which source is a composite interaction. A composite interaction is completed if all sub-interactions have been completed or inhibited. Thus, a synchronization point in form of an interaction $Sync_{c,j}$ has to be added in order to be able to define weather the composite interaction has been completed or not. Thus, the actors executing the communication actions of this interaction have to be the common actors of all final interactions of the composite interaction and the actors executing the interaction that is the target of the relationship in question. It might be that there exists no common actor executing these interaction. In this case there is no actor assigned, which will be discovered in the enforceability algorithm presented later. In lines 6 and 7 new relationship involving this synchronization point are established: $Sync_{c,j}$ is the source of a new relationship of the considered type connecting to the target of the original relationship. Moreover it is the target of weak-precedes relationships from all final interactions of the considered composite interaction. In line 8 all relationships of the considered type which source is a composite interaction are deleted.

The second part of the algorithm deals with relationships targeting a composite interaction. In lines 9 to 11 all precedes- and weak-precedes relationships which target is a composite interaction are substituted. When a composite interaction is enabled, its initial sub-interactions should be enabled. Thus, precedes- and weak-precedes relationships are substituted with relationships of the respective type from the source of the original relationship to the initial interactions of the composite interaction. The last two lines of the algorithm depict the substitution of inhibits relationships which target is a composite interaction. In this case, the new relationships are established to every sub-interaction of the composite interaction.

3.2 Reachability analysis

Well-formed choreographies in Let's Dance may contain *unreachable interactions*, that is, interactions that will never occur in any execution of the choreography. The presence of unreachable interactions makes the analysis of choreographies (e.g. checking for local enforceability) more difficult. Choreographies containing interactions with unreachable interactions are semantically incorrect, and it is thus normal to expect that they should be corrected prior to analysing them further in view of generating local models. Three patterns lead to an interaction i being unreachable.

1. Two interactions with vice-versa inhibits relations precede i (Figure 6(a)).
2. An interaction preceding i, inhibits i (Figure 6(b)).
3. An interaction j that always executes inhibits i and there is also a path of precedes and weak-precedes relations from that interaction to i (Figure 6(c)).

```
1: $Precedes := Precedes \cup \{(i,j) \in I \times I \mid \exists k \in CI[i\ Precedes\ k \wedge k\ Precedes^{*}\ j]\}$
2: **for each** $R \in \{Precedes, WeakPrecedes, Inhibits\}$
3:     **for each** $(c,j) \in R$ **where** $c \in CI$
4:         $I := I \cup \{Sync_{c,j}\}$ (*this creates a new interaction $Sync_{c,j}$*)
5:         $Performs(Sync_{c,j}) := \{a \in Actors \mid (\forall f \in Final(c)[a \in Performs(f)]) \wedge a \in Performs(j)\}$
6:         $R := R \cup \{(Sync_{c,j}, j) \mid c\ R\ j\}$
7:         $WeakPrecedes := WeakPrecedes \cup \{(i, Sync_{c,j}) \mid i \in Final(c)\}$
8:     $R := R \setminus (CI \times I)$
9: **for each** $R \in \{Precedes, WeakPrecedes\}$
10:    $R := R \cup \{(i,j) \in I \times I \mid \exists a \in Ancestors(j) \wedge j \in Initial(a) \wedge i\ R\ a\}$
11:    $R := R \setminus (I \times CI)$
12: $Inhibits := Inhibits \cup \{(i,j) \in I \times I \mid \exists a \in Ancestors(j)[i\ Inhibits\ a]\}$
13: $Inhibits := Inhibits \setminus (I \times CI)$
```

Figure 5. Algorithm for expanding relationships involving composite interactions

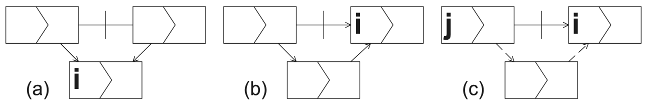

Figure 6. Unreachable Interactions

An algorithm for detecting unreachable interactions is presented in Figure 7. This algorithm takes as input an expanded choreography (i.e. pre-processed by the expansion algorithm in Figure 5), and produces a set of unreachable interactions (namely U). The algorithm makes use of the following auxiliary relations:

- $Prec = Precedes \cup WeakPrecedes$
- $Inhibits'$: a variant of $Inhibits$ that excludes certain relationships that do not have any effect. Such relationships are characterized by the fact that the target interaction can only occur after the source interaction has completed and thus the source never manages to actually inhibit the target (e.g. two interactions being connected with a precedes relationship in one direction and with a inhibits relationship in the opposite direction). Thus: $Inhibits' = Inhibits \setminus (Prec^{-1})^{*}$.

Lines 2-5 of the algorithm take care of patterns 1 and 2 identified above. Pattern 3 is more difficult to detect since there is the condition that interaction j has to always execute. Preceding guarded interactions (line 7) and targets of inhibits relations might prevent this (line 8). However, there are two cases where inhibits relations do not have any effect: (i) if the target always happens before the source (which we address by using $Inhibits'$ instead of $Inhibits$); or (ii) if the source is unreachable. To detect this latter case we proceed in two steps. For each interaction i, if there is an inhibits relationship targeting a preceding interaction j, we temporarily classify i as unreachable but we keep track that this classification is subject to revision. We do this by inserting a tuple into an auxiliary relation called $Depends$, indicating that the reachability of i "depends" on the reachability of the source of the inhibits relation targeting j (lines 12-13). Indeed, if the source of this inhibits relation turns out to be itself unreachable, and thus will never be executed, the inibits relationship in question will never impede the execution of j. In a second step, after having fully populated the relation $Depends$, if it is found that all the interactions upon which the reachability of i depends have been classified as unreachable, i and all its successors are definitely classified as unreachable as well (lines 15-16). Otherwise i is classified as reachable (line 18).

In the rest of the paper, we restrict ourselves to choreographies without unreachable interactions, that is $U = \emptyset$.

3.3 Enforceability algorithm

The algorithm for checking local enforceability is presented in Figure 8. This algorithm implements a function that takes as parameter an expanded global model (i.e. a global model after applying the algorithm in Figure 5) and produces a set of pairs of interactions (i, j) such that there is a non-locally enforceable constraint between i and j. This set of pairs is named *UR*. Moreover, the algorithm produces as set of interaction *UI* that include guarded and repeated interactions which repetition instructions and guards respectively can not be enforced.

In lines 1-3 of the algorithm, each relationship is checked to ensure that there is at least one common actor involved both in the source and in the target interaction. If this is not the case, the respective pair of interactions is added to the set of relationships that are not locally enforceable. The "for each" loop starting in line 4 deals with more complex requirements for weak-precedes relationships between pairs of interactions (i, j). The "for each"-loop from line 5 to line 8 checks each inhibits relationships that could cause the source of the weak-precedes relationship (i) to be skipped. The first part of the condition in the if-clause can only evaluate to true, if there is no direct or transitive precedes relationship between the interaction that can be inhibited (l) and interaction j, since in this case the enforceability is al-

```
1:  U := {}; Depends := {};
2:  **for each** $(i, j) \in Precedes$ **sorted topologically**
3:      **if** $(i \in U \vee \exists x \in I\ [x\ Inhibits\ j \wedge x\ Precedes^*\ i]$
4:          $\vee \exists x, y \in I\ [x\ Inhibits\ y \wedge y\ Inhibits\ x \wedge x\ Precedes^*\ i \wedge y\ Precedes^*\ j])$ **then**
5:          $U := U \cup \{j\}$
6:  **for each** $(i, j) \in Inhibits$
7:      **if** $\exists (x, y) \in WeakPrecedes\ [i\ Prec^*\ x \wedge y\ Prec^*\ j \wedge \neg \exists w \in GI(w\ Precedes^*\ x)]$ **then**
8:          **if** $\neg \exists (v, w) \in Inhibits'\ [w\ Precedes^*\ i]$ **then**
9:              $U := U \cup \{k \in I \mid j\ Precedes^*\ k\}$
10:         **else**
11:             $U := U \cup \{j\}$
12:             **for each** $(v, w) \in Inhibits'$ **where** $w\ Precedes^*\ i$
13:                 $Depends := Depends \cup \{(j, v)\}$
14: **for each** $i \in I$ **where** $\exists j \in I(i\ Depends\ j)$
15:     **if** $\neg \exists j \in I \setminus U(i\ Depends^+\ j \wedge \neg \exists k \in U(k\ Precedes^+\ j))$ **then**
16:         $U := U \cup \{k \mid i\ Precedes^*\ k\}$
17:     **else**
18:         $U := U \setminus \{i\}$
```

Figure 7. Algorithm for detecting unreachable interactions

ready ensured. The second part of the condition that has to be fulfilled in order to add the pair of interactions (i, j) to the set of interactions with a non-locally enforceable constraint starts after the $\wedge$-symbol in line 6 and ends with line 7. It evaluates to true if there exists an actor that is involved in the execution of interaction j and in an interaction that is involved in a path of consecutive precedes relationships going from the interaction that might be inhibited (l) to the source of the weak-precedes relationship (i), and this actor not being involved in the execution of an interaction following the source of the inhibits relationship (k). This additional condition formulates the necessity, that any actor that is involved in executing interaction j and that is involved in the execution of an interaction that might be skipped due to the inhibits relationship, must have knowledge about the result of the evaluation of the guard. Therefore, the interaction that provides this knowledge to the respective actor (n), must be executed in any possible instance of the choreography, which is expressed by the relating this interaction with a $Heralds$ relationship. We say that an interaction x "heralds" another interaction y if in any run where interaction x occurs, interaction y necessarily occurs subsequently. If the above "heralds" relationship holds between j and some suitable n, holds, it can be assured that at least one of the actors involved in j will know that i has been skipped (since they would know that an alternative path was taken when interaction n eventually occurs), and thus the actor(s) in question would know that they can complete interaction i. In this case, the weak-precedes relationship between i and j is enforceable, otherwise it may not be enforceable and the pair (i, j) is added to UR. An interaction may be skipped either due to the presence of guards or due to inhibits relationships, hence the relation $Heralds$ is defined as follows:

$$
\begin{aligned}
Precedes^* \setminus \{(i, j) \in I \times I \mid \exists z \in I\ [i\ Precedes^+\ z \wedge\ \\
z\ Precedes^*\ j\ \wedge (z \in GI \vee \exists x, y \in I\ \\
[x\ Inhibits'\ y\ \wedge y\ Precedes^*\ z]]\}
\end{aligned}
$$

The "for each" loop in lines 9-12 checks each guarded interaction that directly or transitively precedes i. If the guard of one such interaction evaluates to false, the guarded interaction (say k) as well as all interactions directly or transitively connected to it via a precedes relationship, will be skipped. This "path skipping" must be known by at least one of the actors that perform i and j. Thus, there has to be a common actor involved in the execution of i and j, and at least one of these actors must evaluate the guard in question, so as to know whether the path is skipped or not. Additionally, it must be ensured that any actor involved both in j and in an interaction lying on a path of consecutive precedes relationships going from k to i, evaluates the guard. In other words, any actor involved in interaction j and in an interaction that might be skipped if the considered guard evaluates to false, must know the result of the evaluation of the guard. If any of these conditions is not fulfilled, the weak-precedes relationship from i to j is not locally enforceable.

The "for each" loop in lines 13-21 deals with special requirements for repeated interactions. This part of the algorithms makes use of a construct called $Coordinators$. The coordinators is a set of actors that is defined for each repeated and guarded interaction. For elementary interactions, this set contains all actors that are performing a given interaction and that are evaluating the guard or executing the repetition instruction respectively assigned. For composite interactions, this set contains all actors that are involved in the execution of all initial and all final interactions, and that are involved in the evaluation of the guard or the execution of the repetition instruction respectively.

```
1:  for each $(i,j) \in Precedes \cup Inhibits' \cup WeakPrecedes$
2:      if $Performs(i) \cap Performs(j) < 1$
3:      then $UR := UR \cup \{(i,j)\}$
4:  for each $(i,j) \in WeakPrecedes$
5:      for each $k,l,m,n \in I$ where $k\ Inhibits'\ l \wedge l\ Precedes^*\ i$
6:          if $\neg(l\ Precedes^+\ j) \wedge (\exists a \in Performs(j) \exists m \in I \neg \exists n \in I[l\ Precedes^*\ m \wedge m\ Precedes^*\ i \wedge$
7:          $a \in Performs(m) \wedge a \in Performs(n) \wedge k\ Heralds\ n]$
8:          then $UR := UR \cup \{(i,j)\}$
9:      for each $k \in GI$ where $k\ Precedes^*\ i$
10:         if $\neg(k\ Precedes^+\ j) \wedge (Performs(j) \cap Evaluates(k) \cap Performs(i) = \emptyset \vee \exists a \in Performs(j) \exists m \in I$
11:         $[k\ Precedes^*\ m \wedge m\ Precedes^*\ i \wedge a \in Performs(m) \wedge a \notin Evaluates(k)])$
12:         then $UR := UR \cup \{(i,j)\}$
13: for each $i \in (RI \cup GI)$
14:     if $Coordinators(i) = \emptyset$
15:     then $UI := UI \cup \{i\}$
16:     for each $(k,j) \in Precedes \cup WeakPrecedes \cup Inhibits'$ where $i \notin Ancestors(j) \wedge i \in Ancestors(k)$
17:         if $Performs(j) \cap Performs(k) \cap Coordinators(c) = \emptyset$
18:         then $UR := UR \cup \{(k,j)\}$
19:     for each $(j,k) \in Precedes \cup WeakPrecedes \cup Inhibits'$ where $i \notin Ancestors(j) \wedge i \in Ancestors(k)$
20:         if $Performs(j) \cap Performs(k) \cap Coordinators(c) = \emptyset$
21:         then $UR := UR \cup \{(j,k)\}$
```

Figure 8. Algorithm for determining local enforceability

$Coordinators(i) :=$ **if** $i \in EI$
then $\{a \in Actors \mid \exists e \in E[(i,e) \in Assignments \wedge$
$a \in Performs(i) \wedge a \in Conducts(e)]$
else $\{a \in Actors \mid \exists e \in E, \exists x \in I,$
$\exists y \in I[(i,e) \in Assignments \wedge x \in Initial(i) \wedge$
$y \in Final(i) \wedge a \in \{Performs(x) \cap Perfomrs(y)$
$\cap Conducts(e)\}]\}$

In lines 14 and 15 the set of coordinators is checked. If this set is empty, the respective interaction is added to the set of interactions that are not locally enforceable. The foreach loop from line 16 to 18 iterates over all relationships, where a sub-interaction of the considered interaction is the source of a relationship to an interaction that is not a sub-interaction of the considered interaction. For such relationships to be enforceable, there must exist an actor involved in the interactions j and k that is part of the set of coordinators of the considered interaction. In all other cases, there exists no actor that is able enforce the relationship in question. Thus, the relationship would be added to the set of relationship that are not enforceable. The last three lines of the algorithm are very similar to the ones before: the only difference is that in this case all relationships are examined, where the target of a relationship is a sub-interaction of the interaction in question and the source is not.

4 Generating Local Models

The aim of the algorithm for generating local models is to provide a local view for each of the actors participating in a choreography. These local views should only include elementary interactions where at least one of the communication actions is executed by the actor for which the local view is generated. The challenge in generating local views is the derivation of the correct relationships between interactions. We illustrate this issue using the global view depicted in Figure 9. There are three participating actors in this global model: “a1” playing the role “A”, “b1” playing the role “B” and “c1” playing the role “C”.

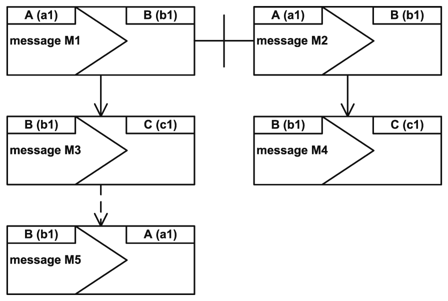

Figure 9. Sample global model

The local view for actor “b1” equals to the whole choreography since this actor participates in every interaction. Meanwhile, the local model of actor “a1” consists of the two interactions labeled “M1” and “M2” related via a two-way inhibits relationship, plus the elementary interaction labeled “M5”. The latter interaction is not directly related with “M1” in the global model, yet, in order to preserve the semantics of the global model, the local model of “a1” needs to explicitly render the “derived” weak-precedes relationship between “M1” and “M5” (see the local model of actor “a1” in Figure 10). Finally, the local model of “c1” consists of the two interactions labeled “M3” and “M4”. These interactions are unrelated in the choreography, yet, in the local model of “c1” they are related via a two-way inhibits relationship (see Figure 10), since only one of these two interactions can ever occur.

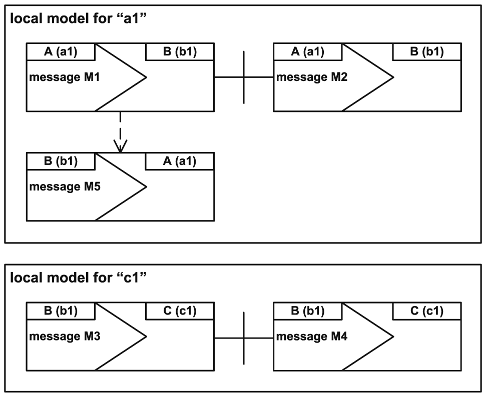

Figure 10. Generated local models

The algorithm in Figure 11 describes the generation of the local model for a given actor in a choreography. Following the idea of abstracting from certain process steps in process algebra by viewing them as “silent” or τ-steps and then reducing the process to an equivalent one without τ-steps [3], the algorithm proceeds by viewing elementary interactions in which no communication action is executed by a considered actor as a silent or τ-interaction. A set of reduction rules are then applied to remove these τ-interactions. The algorithm presented below implements a procedure that, given a global model and an actor *a*, computes the local model corresponding to the view of *a* on the assigned global model. The “for each”-loop in line 2 iterates over every τ-interaction in the considered local view and tries to apply the six reduction rules to this τ-interaction. The reduction rules can be integrated in three groups, which equals to the three “for each”-loops starting in lines 3, 10, and 17: the first one for τ-interactions that are the target of a precedes relationship, the second one for τ-interactions that are the target of a weak-precedes relationship and the third one for τ-interactions that are the source of a precedes relationship. Each of these three loops consists of two further for-each loops, whereby each of them represents the second relationship of a combination of two relationships to be reduced. Only with this structure it is possible to consider every possible combination of relationships and therewith not miss any possible reduction.

The first reduction rule applies for every combination of two consecutive precedes relationships where the considered τ-interaction is in between. From this structure a precedes relationship between the source of the first and the target of the second precedes relationship can be derived.

```
 1: Procedure LocalModel(a: Actor)
 2:   for each $\tau \in I$ where $\neg a \in Performs(\tau)$
 3:     for each $i \in I$ where $i\ Precedes\ \tau$
 4:       for each $j \in I$ where $\tau\ Precedes\ j$
 5:         $Precedes := Precedes \cup \{(i, j)\}$
 6:         if $\exists e \in E[(\tau, e) \in Assigned]$
 7:         then $Assigned := Assigned \cup \{(j, e)\}$
 8:       for each $j \in I$ where $\tau$ WeakPrecedes $j$
 9:         WeakPrecedes := WeakPrecedes $\cup\{(i, j)\}$
10:     for each $i \in I$ where $i$ WeakPrecedes $\tau$
11:       for each $j \in I$ where $\tau$ WeakPrecedes $j$
12:         WeakPrecedes := WeakPrecedes $\cup\{(i, j)\}$
13:       for each $j \in I$ where $\tau\ Precedes\ j$
14:         WeakPrecedes := WeakPrecedes $\cup\{(i, j)\}$
15:         if $\exists e \in E[(\tau, e) \in Assigned]$
16:         then $Assigned := Assigned \cup \{(j, e)\}$
17:     for each $i \in I$ where $\tau\ Precedes\ i$
18:       for each $j \in I$ where $i\ Inhibits'\ \tau$
19:         $Inhibits' := Inhibits' \cup \{(j, i)\}$
20:         if $\exists e \in E[(\tau, e) \in Assigned]$
21:         then $Assigned := Assigned \cup \{(i, e)\}$
22:       for each $j \in I$ where $\tau\ Inhibits'\ j$
23:         $Inhibits' := Inhibits' \cup \{(i, j)\}$
24:         if $\exists e \in E[(\tau, e) \in Assigned]$
25:         then $Assigned := Assigned \cup \{(i, e)\}$
26:     for each $i \in I$
27:       for each $R \in \{Precedes, WeakPrecedes,$
28:                       $Inhibits'\}$
29:         $R := R \setminus \{(i, \tau), (\tau, i)\}$
30:     $I := I \setminus \{\tau\}$
```

Figure 11. Algorithm for deriving local models from choreographies

Additionally, if the τ-interaction has a guard or a repetition instruction assigned, then this expression is assigned to the interaction that is the target of the second precedes relationship. This pushing of expressions to following interactions is executed in four of the six reduction rules and might lead to expressions that are assigned to interactions, which execution do not involve the actors nominated for evaluating this expression. In this case, the expression is considered as opaque and analogously considered while generating code-fragments. The second rule applies for a weak-precedes relationship as second relationship. In this case a weak-precedes relationship between the source of the precedes relationship and the target of the weak-precedes relationship can be derived.

The next two reduction rules apply when a τ-interaction is the target of a weak-precedes relationship: in the first rule with the τ-interaction being the source of an weak-precedes relationship, in the second rule with the τ-interaction being the source of a precedes relationship. In both cases a weak-precedes relationship is derived, connecting the source of the original weak-precedes relationship and the target of the second original relationship. In the latter case any expression assigned to the τ-interaction is assigned to the interaction that is the target of the precedes relationship. The last two reduction rules apply for combinations of one precedes and one inhibits relationship. The first one covers the cases where the τ-interaction is the source of a precedes relationship and the target of an inhibits relationship. This leads to a derived inhibits relationship where the source of the original inhibits relationship is the source and the target of the original precedes relationship is the target of the derived relationship. The last reduction rule covers all combinations of relationships where the considered τ-interaction is the source of a precedes and of an inhibits relationship. In this case an inhibits relationship can be derived, having the target of the original precedes relationship as source and the target of the original inhibits relationship as target. Both reduction rules involving inhibits-relationships have the pushing down of expressions to the interaction that is the target of the precedes-relationship incorporated. Having applied all reduction rules, the considered τ-interaction and all relationships in which it is involved are deleted (lines 18-22).

5 Related Work

The description of global service interaction models (also called choreographies) has been the subject of intensive research and standardization. This had led to various special-purpose languages such as WS-CDL, BPSS/ebBP and EDOC (see references in Section 1). In contrast, the issue of local enforceability of global models has received little attention. Existing formalizations of WS-CDL (e.g. [16]) skirt the issue of local enforceability of choreographies. Instead, they assume the existence of a state (i.e. a set of variables) shared by all participants. Participants synchronize with one another to maintain the shared state up-to-date. This effectively means that some interactions take place between services to synchronize their local view on the shared state, and at least some of these interactions are not explicitly defined in the choreography. In the worst case, this could lead to situations where a business analyst signs off on a choreography, and later it turns out that in order to properly execute this choreography a service provided by one organization must interact with a service provided by a competitor, unknowingly of the business analyst. A similar approach is followed in [5], where the authors formalize relationships between choreographies and orchestrations (i.e. local models). Meanwhile, in defining a translation from WS-CDL to BPEL, [11] adopts a different approach: control dependencies between interactions that can not be enforced in the local models are simply ignored. In this paper, we argue that neither approach (introducing hidden interactions or ignoring unenforceable dependencies) is satisfactory. Instead, we propose tool support as an approach to detect unenforceability of choreographies.

Let's Dance draws upon previous work in the area of workflow and architecture description languages. In particular, the "weak precedes" construct is inspired from (though not equivalent to) the "weak sync" construct defined in the ADEPT workflow definition language [14]. Meanwhile, the "inhibits" construct is inspired from the "disabling" construct of the Interactive Systems Description Language (ISDL): an architecture description language that supports the specification of behavioral dependencies between component interactions. A discussion on the application of ISDL for service choreography modeling is presented in [13]. However, the suitability of ISDL for capturing complex service interactions (e.g. involving multicast) is unproven.

In [4], the authors consider the use of state machines for describing local models of service interactions. While state machines lead to simple models for highly sequential scenarios, they may lead to spaghetti-like models when used to capture scenarios with parallelism and cancellation. Nonetheless, state machines have been shown to be a suitable formal foundation for reasoning about service models [9]. This latter reference surveys a number of approaches for describing service interaction models based on communicating state machines. None of the proposals covered by this survey addresses the issue of local enforceability of global models. Instead, service interactions are described as collections of interconnected local models.

Foster et al [8] suggest the use of Message Sequence Charts (MSCs) for describing global service interaction models. These global models are converted into local models expressed as FSMs for analysis purposes. It should be noted that MSCs are a notation for describing behavior scenarios as opposed to full behavior specifications. In particular, basic MSCs do not allow one to capture conditional branches, parallel branches, and iterations. Extensions to MSCs to capture complex behavior have been defined, but in realistic cases they lead to cluttered diagrams since MSCs are based on lifelines which are fundamentally targeted at capturing sequencing rather than branching.

6 Conclusion and future research directions

This paper has put forward the issue of local enforceability of global service interaction models (choreographies). An algorithm is proposed that analyzes the relationships between interactions defined in a choreography and identifies those that are not enforceable. A second algorithm serves the purpose of generating local models for each actor participating in a given choreography. These local models can be used as a basis for generating behavioral interfaces and executable service descriptions. In a separate work we have formalized the control-flow constructs of the language using π-calculus [7].

We are implementing a toolset that supports the modeling of Let's Dance choreographies and the development of services based on these models. The algorithms for choreography expansion, reachability analysis and local enforceability have been implemented. Ongoing work aims at implementing the algorithm for local models generation. Future work will aim at further validating the proposed language by designing and implementing model transformations for generating code templates (e.g. in BPEL) from local models. Also, we are working on extending the formal semantics with respect to communication and correlation.

References

[1] 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*, May 2003. Available at: `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`

[2] A. Barros, M. Dumas, and A. H.M. ter Hofstede: *Service Interactions Patterns*. In Proceedings of the 3rd International Conference on Business Process Management (BPM), Nancy, France, September 2005. Springer Verlag, pp. 302-218.

[3] J. C. M. Baeten, W. P. Weijland: *Process Algebra*, Cambridge University Press, 1990.

[4] B. Benatallah, F. Casati, F. Toumani, and R. Hamadi: *Conceptual Modelling of Web Service Conversations*. In Proceedings of 15th International Conference on Advanced Information Systems Engineering (CAiSE'03), Velden, Austria, June 2003, pp. 449-467 Springer Verlag.

[5] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro: *Choreography and Orchestration Conformance for System Design*. In Proceedings of 8th International Conference on Coordination Models and Languages (COORDINATION'06), Bologna, Italy, June 2006, Springer Verlag.

[6] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, K. Riemer (Eds). *ebXML Business Process Specification Schema Version 1.01*, UN/CEFACT and OASIS Specification, May 2001. `http://www.ebxml.org/specs/ebBPSS.pdf`.

[7] G. Decker, J.M. Zaha, M. Dumas: *Execution Semantics for Service Choreographies*. In Proceedings of 3rd International Workshop on Web Services and Formal Methods (WS-FM 2006), Vienna, Austria, September 2006. LNCS.

[8] H. Foster, S. Uchitel, J. Magee, J. Kramer: *Tool Support for Model-Based Engineering of Web Service Compositions*. In Proceedings of the IEEE International Conference on Web Servies (ICWS), Orlando FL, USA, July 2005. IEEE Computer Society.

[9] R. Hull, J. Su: *Tools for composite web services: a short overview*. SIGMOD Record 34(2): 86-95, 2005.

[10] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. *Web Services Choreography Description Language Version 1.0*, W3C Candidate Recommendation, November 2005. `http://www.w3.org/TR/ws-cdl-10`.

[11] J. Mendling, M. Hafner: *From Inter-organizational Workflows to Process Execution: Generating BPEL from WS-CDL*. In Proceedings of the OTM Workshops 2005, Agia Napa, Cyprus, November 2005, pp. 506-515, Springer Verlag.

[12] Object Mangement Group (OMG): *UML Profile for EDOC*. `http://www.omg.org/technology/documents/formal/edoc.htm`

[13] D.A.C. Quartel, R.M. Dijkman, M. van Sinderen: *Methodological support for service-oriented design with ISDL*. In Proceedings of the 2nd Intenational Conference on Service-Oriented Computing (ICSOC), New York NY, USA, November 2004, pp 1–10, Springer Verlag.

[14] M. Reichert, P. Dadam: *ADEPTflex – Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems 10(2): 93-129, 1998.

[15] S. White: *Business Process Modeling Notation (BPMN)* – Version 1.0, May 3 2004, `http://www.bpmi.org`.

[16] H. Yang, X. Zhao, Z. Qiu, G. Pu , and S. Wang: *A Formal Model for Web Service Choreography Description Language (WS-CDL)* Preprint, School of Mathematical Sciences, Peking University, January 2006 `http://www.math.pku.edu.cn:8000/var/preprint/7021.pdf`

[17] J.M. Zaha, A. Barros, M. Dumas, A. ter Hofstede: *Let's Dance: A Language for Service Behavior Modeling*. Technical Report FIT-2006, Faculty of IT, Queensland University of Technology. `http://eprints.qut.edu.au/archive/00004468/`