Automatic Skeleton Generation for Data-Aware Service Choreographies

Huu Nghia Nguyen*, Pascal Poizat** and Fatiha Zaïdi*

* LRI; Univ. Paris-Sud, CNRS, Orsay, France
** LIP6; Univ. Paris Ouest Nanterre La Défense, CNRS, La Défense, France
huu-nghia.nguyen@lri.fr, pascal.poizat@lip6.fr, fatiha.zaidi@lri.fr

***Abstract*—Service-oriented engineering is an emerging software development paradigm for distributed collaborative applications. Services are developed independently and are composed to achieve common requirements. Service choreographies define such requirements from a global perspective, based on interactions among a set of participants that are implemented as services. In this paper, we support a reliable data-aware service choreography development process through a dedicated projection. It extracts, from a choreography, a behavioral skeleton for each of its participants. The projection is valuable in a top-down approach, where developers have only to complete the skeletons with some business code in order to get a distributed application that matches the choreography requirements. The projection is also valuable in a bottom-up approach, where the skeletons can act as controllers between reused services in order to enforce the respect of the choreography. Our approach is supported with a tool that can be downloaded or used online.**

***Keywords*—Service Choreography, Value-Passing Processes, Contract Compliance, Contract Projections, Symbolic Transition Systems, Tool.**

I. Introduction

Service-oriented engineering is an emerging paradigm for the development of distributed applications. Such an application is made up of several entities abstracted as services, each of them being for example a Web application, a Web service, or even a human. To reach a common objective, services have to coordinate by interacting with each other. *Choreographies* support such a collaborative vision of a distributed application. A choreography is the specification of what the collaboration participants, or *roles*, should achieve altogether. Due to its global perspective, a choreography focuses on *interactions* between two roles. A choreography is a constraint that each *service* implementing a role has to follow.

There exist two different modeling approaches for choreographies [1]: interconnected interface models and interaction models. In *interconnected interface models* (*e.g.*, BPMN collaboration diagrams, BPEL4Chor, MSC, UML sequence diagrams, Reo), the basic events are defined at the role level (*e.g.*, sending or receiving a message). Global level interactions are then defined by roughly connecting these events. To the contrary, in *interaction models* (*e.g.*, WS-CDL, BPMN 2.0 choreography diagrams), the basic events are the interactions between roles. This makes interaction models suitable not only for a bottom-up development process – where one has services to reuse and to compose – but also for a top-down development process – where one starts with a global specification of the distributed application to be.

Besides the global perspective, interactions may be seen from the local perspective of each role. From this perspective, only interactions that directly involve the role are captured. Consequently, there exist two kinds of models, local models, or *role models*, which specify the local behaviors of roles (one for each), and global models, or *choreography models*, that correspond to choreography specifications. Choreography models are useful during the early phases of system analysis and design thanks to its global perspective, while role models are blueprints for the implementation of services realizing roles, for the derivation of test cases for these services, and for the adaptation of reused services to the choreography constraints.

Consequently *bridging the gap between choreography and role models* is a cornerstone for top-down choreography development processes. This relates to the projection of relevant role models, or *skeletons*, from a choreography model as shown in Figure 1. To ensure the correctness of this projection, the conformance between the choreography model and the composition of the generated role models should be verified.

A fundamental issue of choreography, called *realizability*, is whether a choreography model can be correctly projected to role models. Generally, not all choreographies are realizable. Let us take the two unrealizable choreographies described in Figure 2. The left hand side choreography describes, in the BPMN 2.0 choreography notation, a `request` interaction between roles *buyer* and *vendor* followed by a `ship` interaction between roles *warehouse* and *shipper*. This choreography is not realizable since role *warehouse* has no possibility to know when `ship` must be done. It is after `request`, but the

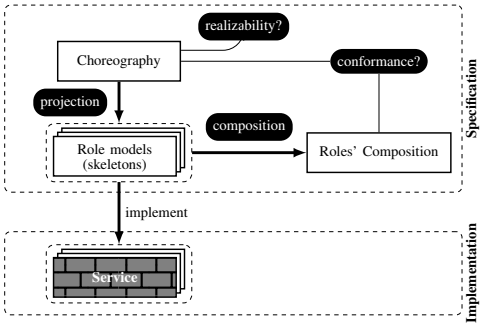

Fig. 1. Top-down approach of choreography development

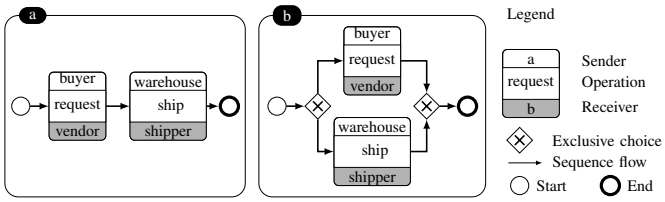

Fig. 2. Examples of unrealizable choreographies

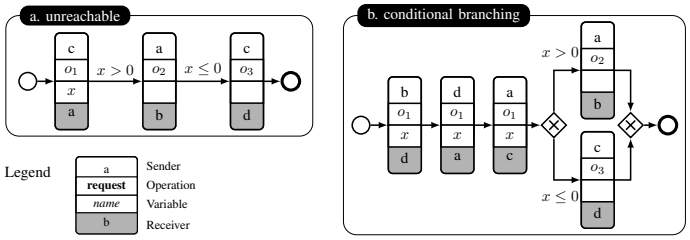

Fig. 3. Impact of data: unrealizability → realizability

$warehouse$ never knows when `request` occurs. The right hand side choreography requires that either `request` or `ship` is done but there is no way to ensure that since different sender roles, *i.e.*, the $buyer$ and the $warehouse$, are concerned.

The rest of the paper is organized as follows. We first motivate our work by presenting in Section II the impact of value-passing on choreography models and on realizability, together with details on possible application domains. In this section, we also give an overview of our work and its contributions. We then introduce in Section III our formal model for data-awareness interaction-based service choreographies. A projection from choreography to local model of each role is defined in Section IV. The projection correctness is ensured in Section V by analyzing the choreography realizability. The implementation of our framework and experiments are discussed in Section VI. After presenting related work in Section VII, we conclude and discuss perspectives of our work.

II. Motivation and Overview

In this section we motivate our work by firstly pointing to the lack of data-aware support in interaction-based service choreography. We then introduce some applicable domains which can directly reuse the results of this paper. Finally, we present our contribution and give an overview of our approach.

A. Issue: Choreography with Value-Passing

Most existing approaches do not adequately support the value-passing, *i.e.*, without explicitly considering the data exchanged through interactions and using it for branching decisions. They just *abstract away from data*. This may yield *over-approximation issues*, *e.g.*, false negatives in verification process. Hence the presence of value-passing in choreography model may change its realizability property. Let us take an extension of the unrealizable choreographies in Figure 2 as depicted in Figure 3, which becomes realizable thanks to the value-passing. In this figure, the BPMN 2.0 notations are extended to support data [2], hence an interaction is attached with a variable. The left choreography describes that firstly there is an interaction o_1 from role a to role c with data exchange called x. Then the interaction o_2 from a to b should be done if $x > 0$. Finally, the interaction o_3 from c to d should be done if $x \leq 0$. It is straightforward to see that the interaction o_3 is never executed. Hence there is no need of o_3 in the choreography. The choreography becomes realizable when only o_1 and o_2 are required to be implemented. The right choreography is realizable as well, when the communication is synchronous or on an asynchronous sending mode. After o_1 has been done, all roles know x so they can decide to do (send or receive) o_2 or o_3.

It is critical to support value-passing in modeling and then in analyzing service choreographies. Some approaches do this by working on *closed implementation-level systems*, *i.e.*, value-passing is only with ground data. In such a case, this could lead into serious efficiency problems, *state space explosion*, when analyzing choreography. Some others avoid this problem by analyzing choreography based on the syntax when checking realizability. However, these approaches miss the cases of unreachable and conditional branching as in Figure 3. Moreover, the models used in these works are interaction-based but data are expressed as the way of interconnection-based, *i.e.*, the variables are defined at each local role then their values are synchronized thanks to interactions. This is not adequate for an interaction-based approach. The interactions together with variables should be the basic events. In the choreographies of Figure 3, variable x is attached to the interaction o_1, in which one does not need to specify explicitly that the first condition $x > 0$ must be done at a and the second one $x \leq 0$ at c.

B. Application Domains

1) Generation of Source Code for Service Implementations: In a top-down development process, once the behavioral skeletons have been generated from the choreography with the approach we propose in this paper, one has to implement the business logic behind the protocols expressed in the skeletons. In order to get full compatibility between the business logic and the behavioral protocols, one may use results presented in [3]–[5] to retrieve Java code from skeleton models. This is made possible since the formal models we use for skeletons, Symbolic Transition Graphs, are very close to the Symbolic Transition Systems used in the above-mentioned approaches

2) Generation of Distributed Test Cases: Role models can be used to generate test cases. Many approaches support the production of tests from symbolic transition models. In [6], we generate symbolic test cases according to different coverage criteria or test purposes. A product is performed between the test purpose and the symbolic model. A symbolic execution tree for this product is generated and used to obtain symbolic test cases, *i.e.*, test cases with free variables and constraints on their values. These test cases are then realized and executed against the implementation with the use of a constraint solving tool. Role models generated with the approach we propose yield global conformance with the choreography. In such a case, the interaction part is guaranteed, in other words testing in isolation each service implementing a role can be sufficient to warranty the correction of the choreography implementation.

3) Resolving Mismatch for Service Choreography: Using a bottom-up development process, each peer of a distributed

collaborative application may be independently developed and deployed across different organizations. It is therefore unavoidable that mismatch may arise at both signature and protocol levels, and need to be identified and solved. When two or more peers are incompatible, an adaptor may be introduced to solve mismatch [7]. The adaptor runs parallel with the peers and guides their execution, *e.g.*, avoiding deadlock of their composition. In combination with adaptation, the approach we propose in this paper can be used to resolve mismatch between reused services and between these services' composition and the requirements expressed by the choreography. Skeletons constitute local contracts that services have to fulfill to yield an overall composition being correct wrt. the choreography. Given each of these skeletons, and the corresponding service one wants to reuse for it, we can generate an adaptor between the skeleton and the service. This can be performed in a fully automatic way for some mismatch. For the more powerful adaptation approaches, *i.e.*, for more complex mismatch, the process can be aided [8].

C. Contributions and Overview of the Proposed Approach

The first contribution of the paper relies on the observation of lack of value-passing support in modeling and analyzing service choreographies. Based on this statement, we enrich the interaction-based model by extending our previous formal model [9] to deal with free and bound variables at global and local models. We propose a symbolic framework in which, the data variables are manipulated by using symbols rather than their concrete values. This enables one to model and analyze service choreography in presence of data without suffering from state space explosion and without bounding data domains,

Based on the formal model, we define what is a realizable choreography. In this paper, we do not only check whether the choreography is realizable, but also we propose solutions to render it realizable. For that purpose, we build a projection function, dealing with data, which allows to retrieve local models from a global one. The projection is dedicated to enable the realizability of choreography, *i.e.*, when the choreography is unrealizable, some extra interactions can be automatically introduced. Furthermore, the minimum extra interactions are added in order to obtain minimal implementation and traffic. The projection considers both synchronous and asynchronous communication modes.

The last contribution of this work is the availability of a tool[1]. That is a stand-alone software or a web application that can be used directly in a web browser. We performed experiments on a variety of examples.

The rationale of our approach starts with a Symbolic Transition Graph for the choreography. We then check if it is reachable relying on an SMT solving tool. Once unreachable transitions are removed from the initial model, we produce the role models by applying our projection function. Afterwards, we check the realizability of the composed local models with respect to the choreography. In case of a non realizable choreography, we propose an approach that will compute the needed interactions to add in order to obtain a realizable choreography. Furthermore, to ensure that our projection is well defined, we verify that the composed local models produced by the projection conform to the global model. For that purpose, we use a branching symbolic bismulation relation to establish the conformance.

[1] Our tool is freely available at http://schora.lri.fr

III. A Data-Aware Formal Model for Interaction-based Service Choreographies

In this section, we introduce our formal model for service choreography with value-passing. It is an extension of the one presented in our previous work [9] which introduces free interactions and bound sending events (see below). The symbolic semantics of the model are also introduced. Let us start with the formalization of the basic events.

A. Basic Events of Global and Local Models

In interaction-based model, the basic event in choreography is an interaction. An interaction represents a conversation between two roles. In this section, we will extend interactions with data. We distinguish two kinds of interactions: free interactions and bound interactions.

Let $a, b, c, \ldots \in \mathcal{R}$: be a finite set of roles, and
$o, o_1, \ldots \in \mathcal{O}$: be a finite set of operations,

a *free interaction* represents a communication of value of variable x realized through an operation o from role a to b, which is denoted by $o^{[a,b]}.x$, while the bound one is denoted by $o^{[a,b]}.\langle x\rangle$.

The difference between free and bound interaction relies on how their variables representing data exchange are interpreted. In free interaction, the data exchange must be known before the interaction may occur. While in bound interaction, the data exchange is bounded when the interaction occurs. We remark that x may be a complex structure variable. The variable x may be omitted when the interaction does not carry data, *e.g.*, $\mathtt{ACK}^{[a,b]}$, or when it is never used.

At local view, we introduce a (bound) reception $o^{[a,b]}?\langle x\rangle$, a free sending $o^{[a,b]}!x$, and a bound sending $o^{[a,b]}!\langle x\rangle$. They are used to represent reception or sending activities of each participant in choreography. Table I list our basic events, in which we define $\mathtt{fv}(\alpha)$ and $\mathtt{bv}(\alpha)$ as the set of free and bound variables in α. In the Table I, τ event is considered for local model. In the literature, τ events are usually used to represent unobservable events, *e.g.*, internal events. Since we are interested in an abstract formal choreography model, *i.e.*, implementation independent and in interaction-based model, the basic events are interactions, the internal event can be ignored at choreography level [10]. Although there are no internal events in our approach, τ is used to represent an unobservable interaction. Consequently, it does not exist at global model but may appear on local models to express interactions of third participants, *e.g.*, these interactions do not concern the current participant, this is why they are not observable.

B. Symbolic Transition Graphs

In this paper, we formally modeled service choreographies with Symbolic Transition Graphs (STG) [11], which can be used to specify either global view or local view with global or local events. We select STG as model for service choreography since it supports data, guard and free/bound variables [11]–[13].

TABLE I. THE BASIC EVENTS

α	Name	Local/Global	Free/Bound	$\texttt{fv}(\alpha)$	$\texttt{bv}(\alpha)$
$o^{[a,b]}.x$	Free Interaction	global model	f	$\{x\}$	$\emptyset$
$o^{[a,b]}.\langle x\rangle$	Bound Interaction		b	$\emptyset$	$\{x\}$
τ	Silent Event	local model	f	$\emptyset$	$\emptyset$
$o^{[a,b]}?\langle x\rangle$	(Bound) Reception		b	$\emptyset$	$\{x\}$
$o^{[a,b]}!x$	Free Sending		f	$\{x\}$	$\emptyset$
$o^{[a,b]}!\langle x\rangle$	Bound Sending		b	$\emptyset$	$\{x\}$

Many formal approaches and notably conformance relation are based on such a model [14]. Moreover, Symbolic Transition systems, which is very close to STG, are widely used in different areas, *e.g.*, for testing purpose [6], [15], or for code generation [3]–[5]. A STG is a transition system. Each transition of STG is labelled by a guard ϕ and a basic event α. The guard ϕ is a boolean equation which has to hold for the transition to take place. A symbolic transition from state s to state s' with a guard ϕ, and an event α is denoted as $s \xrightarrow{[\phi]\,\alpha} s'$. The guard ϕ of a transition can be omitted if it is true, *e.g.*, $s \xrightarrow{\alpha} s'$.

Formally, a STG is a tuple (S, s_0, T) where, S is a non empty set of finite states, each state s having an associated set of free variables $\texttt{fv}(s)$ which are used by guards or events in next transitions, $s_0 \in S$ is the initial state, and T is a set of finite transitions. If $s \xrightarrow{[\phi]\,\alpha} s'$ is a transition of T then $\texttt{fv}(\phi) \cup \texttt{fv}(\alpha) \subseteq \texttt{fv}(s)$ and $\texttt{fv}(s') \subseteq \texttt{fv}(s) \cup \texttt{bv}(\alpha)$.

The symbolic semantics of STG are given with respect to substitution of variables into fresh variables. Each state s is associated with a substitution σ. Under late semantic, for every transition $s \xrightarrow{[\phi]\,\alpha} s'$, the free variable in ϕ and α will be changed by the substitution of s. Moreover, if α is a bound event, *e.g.*, $o^{[a,b]}.\langle x\rangle$, $o^{[a,b]}?\langle x\rangle$, or $o^{[a,b]}!\langle x\rangle$, then the substitution of s' is the one of s by updating a substitution from x to a fresh variable, instead of a value.

C. Choreography Reachability

A STG is a specific directed graph where each transition is guarded by a condition. A transition t is never fired when its guard is always false for any value of variables, *e.g.*, $(x > 0) \wedge (x < 0)$. In such a case, the transition t is *unreachable*, otherwise it is reachable. If a transitions t is unreachable then the transitions, which are only visited through t, are also unreachable. A STG is *reachable* if all its transitions are reachable. It is straightforward to see that the reachable STG obtained from a STG by removing all unreachable transitions has the same behavior with the original one.

Particularly, given a STG (S, s_0, T), we start the traversal from the initial state s_0. For each outgoing transition t, we verify the conjunction of its guard and its precedent guards accumulating from s_0. If the conjunction is always *false*, then the transition is removed. Furthermore, all transitions which are only visited through t are also removed.

D. Service Choreography Example

In order to illustrate the problems related to the realizability checking of service choreography, we consider an Online Shopping Process (OSP) case study. We model the scenario using an extended BPMN 2.0 choreography [2] as in Figure 4(a). It describes the collaborations between four independent

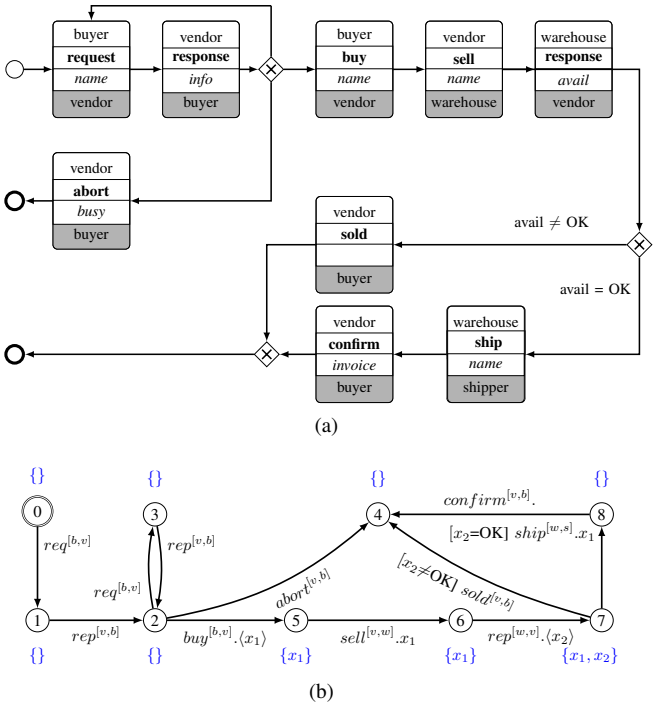

Fig. 4. Online Shopping Process in (a) Extending BPMN 2.0 Choreography [2] and in (b) Symbolic Transition Graph

participants: a *buyer*, a *vendor*, a *warehouse*, and a *shipper*. First, the *buyer* asks the *vendor* for an interested product by indicating the `name` of a requested product. The *buyer* responds with the information of the product `info`. This is repeated until the *buyer* decides to buy a product or it is aborted by the *vendor*. After receiving the buy request, the vendor issues a `sell` command to the *warehouse*. The *warehouse* replies with the status of the product. If the product is available, the *warehouse* transfers the product to the *shipper*, then a confirmation will be issued from the *vendor* to the *buyer*. Otherwise, the product is sold, the *vendor* will notify a `sold` response to the *buyer*.

The STG corresponding to this online shopping choreography is presented in Figure 4(b). For simplicity, we denote b as the buyer, v as the vendor, w as the warehouse, and s as the shipper. The operation and variable names are also reduced. Since the information brought by `name` of the first interaction, `info`, `busy`, and `invoice` do not help any interactions, *i.e.*, using by guard or free interaction, we remove them in the STG. The `name` in the third interaction is kept in the STG since it will be used by the `sell` and `ship` interactions. Let us note that a specification specifies *what* rather than *how* system should or should not be done. Hence in the running example, Figure 4(a), we do not pay attention to how the buyer selects the product itself, but after that we know the name of the selected one. The interaction `buy` should be described by a bound interaction, *e.g.*, $buy^{[b,v]}.\langle x_1\rangle$. Contrarily, in the interaction `sell`, the product name transferred from the *vendor* to the *warehouse* is the one the *vendor* received from the *buyer*, hence the interaction must be described by a free interaction, *e.g.*, $sell^{[v,w]}.x_1$.

TABLE II. NATURAL PROJECTION

interaction α	on role a	on role b	on role $c \notin \{a, b\}$
$s \xrightarrow{[\phi]\ o^{[a,b]}.x} s'$	$s \xrightarrow{[\phi]\ o^{[a,b]}!x} s'$	$s \xrightarrow{o^{[a,b]}?\langle x\rangle} s'$	$s \xrightarrow{\tau} s'$
$s \xrightarrow{[\phi]\ o^{[a,b]}.\langle x\rangle} s'$	$s \xrightarrow{[\phi]\ o^{[a,b]}!\langle x\rangle} s'$	$s \xrightarrow{o^{[a,b]}?\langle x\rangle} s'$	$s \xrightarrow{\tau} s'$

IV. GLOBAL-TO-LOCAL: BRIDGING THE GAP BETWEEN DESIGN AND IMPLEMENTATION

A trivial implementation of a choreography is a single service which plays all roles of the choreography. In such a case, there is no more realizability issue. However, choreography intends to specify, from a global view, a collaboration of a set of roles. Each role in the choreography is a concrete entity taking part in this collaboration. It should be implemented by a distinguishable, independent service. Consequently, an *implementation* of a choreography should be a set of services where each one implements one of the roles and their composition represents the behaviors required by the choreography. This implementation can be constructed based on a projection and local conformance. The local conformance ensures that a service respects its role. It was studied by our previous work [16]. This section is then dedicated to define the projection.

Basically, *projection* is a procedure which takes as an input a choreography model with n roles and outputs a set of n local models, each one representing the required behaviors of a role in the choreography. A set of such local models can be used as a candidate for implementation. Intuitively, behaviors of a local model are extracted from those of global model, in which a participates, *e.g.*, the free interaction $o^{[a,b]}.x$ is projected onto free sending $o^{[a,b]}!x$ of role a, and on reception $o^{[a,b]}?\langle x\rangle$ on role b. The projection of each transition of STG is defined as in Table II. For instance, the projection of choreography STG in Figure 4(b) on its roles after removing some unused τ transitions is depicted in Figure 5.

V. REALIZABILITY

Once local models are generated, the conformance checking can be used to check if their execution conforms to the interaction constraints specified in the choreography. It is done by checking (global) conformance between the composition of local models and the choreography. The projection is correct if the generated local models exactly preserve the communication requirements of the choreography model. The projection can be ensured if it is only applied on realizable choreographies. However, choreography may often not be realizable. It is not very helpful if we simply apply natural projection for realizable choreography and for the rest, we claim them as wrong. The reason of choreography unrealizability is that the interactions involve separate roles. Hence a role cannot compute itself which interactions it must or must not do. The role does the computation when it must perform an interaction after another interaction, or when it has to choose which branch must be followed. Furthermore, when a role uses a free variable, in guard or in free interaction, these variables must be known before by itself in order to validate the guard or to send value of the variable. It is impossible to ensure the choreography realizability without introducing additional interactions.

Going further than determining whether a choreography is realizable, we intend to provide a dedicated projection which

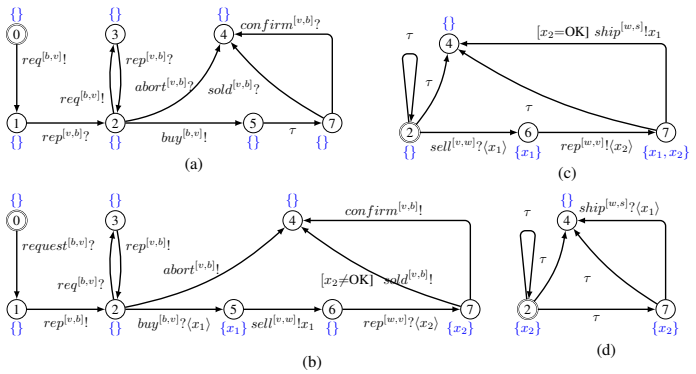

Fig. 5. The projection of STG in Fig. 4(b) on role: (a) buyer, (b) vendor, (c) warehouse, and (d) shipper)

can be used also for unrealizable choreographies. In such a case, the projection is able to propose additional interactions in order to enable the realizability of a choreography. Our projection, which generates local models from a choreography, is performed in several steps. First, we remove all unreachable transitions of the choreography. We then work on a reachable choreography. We calculate a set of additional interactions needed to be added. If this set is not empty, *i.e.*, the choreography is not realizable, then we minimize this set such that there are minimal interactions which are added. After adding the additional interactions to the choreography, we perform the natural projection of the choreography on each role. Finally, we reduce some consecutive transitions which contains τ event. Let us go in detail by analyzing the cases where we need to introduce additional interactions. In the sequel, we examine only reachable STGs, *e.g.*, if a STG is not reachable, we must firstly remove all unreachable transitions.

A. Event connectedness

At global level, an interaction, *e.g.*, α, which is a single event, becomes two separate events: a sending, $\alpha!$, and a reception, $\alpha?$, when it is projected on local models. The causality of the two local events depends on which communication model is considered. In synchronous communication mode, the sending and reception occur at the same time, denoted as $\alpha! = \alpha?$. In asynchronous one, the sending occurs before the reception, denoted as $\alpha! \prec \alpha?$.

The causality of two consecutive interactions, $\alpha_1 \prec \alpha_2$, is considered at local level by the causality of their sending and reception. In synchronous mode, we have $(\alpha_1! \prec \alpha_2!) \vee (\alpha_1? \prec \alpha_2?) \vee (\alpha_1! \prec \alpha_2?) \vee (\alpha_1? \prec \alpha_2!)$. In asynchronous mode, based on the results of [17], [18], we consider three possibilities: $\alpha_1! \prec \alpha_2!$ (sending), $\alpha_1? \prec \alpha_2?$ (reception), and $\alpha_1? \prec \alpha_2!$ (disjoint). To ensure such causality of two consecutive interactions, α_1 and α_2, the relations between their sender, s_1 and s_2 resp., and their receiver, r_1 and r_2 resp., must satisfy one of the conditions below, depending on the communication mode:

- synchronous mode: $\{s_1, r_1\} \cap \{s_2, r_2\} \neq \emptyset$
- sending mode: $s_2 = s_1 \vee s_2 = r_1$
- reception mode: $r_2 = r_1 \vee s_2 = r_1$
- disjoint mode: $s_2 = r_1$

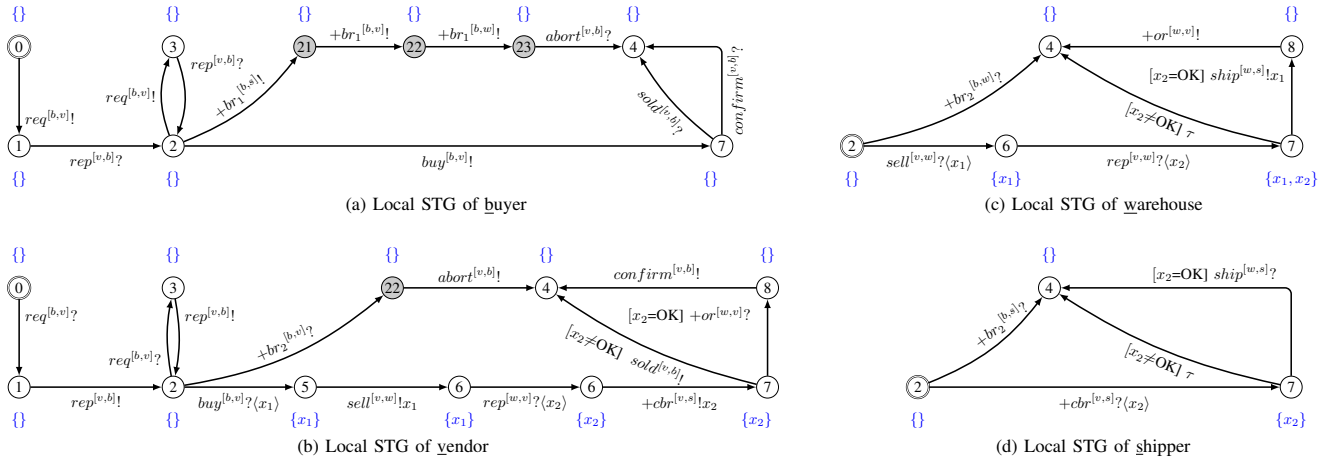

Fig. 6. Projection with additional interactions of STG in Fig. 4(b) under synchronization communication mode

These conditions enable all the participants to compute when they may or may not do the next interaction (send or receive). Let us go into details, for instance, with the sending mode assumption. This mode requires that the sending of the next interaction must happen after the one of the current interaction. If these two interactions have the same sender, *e.g.*, $s_2 = s_1$, then the sender always enables this condition. Moreover, if the sender of the next interaction and the receiver of the current one are the same, *e.g.*, $s_2 = r_1$, when it receives the current interaction, it knows that the sending of the current interaction has already occurred, and thus it may enforce the condition by sending the second interaction.

If two consecutive transitions, *e.g.*, $s \xrightarrow{[\phi_1]\,\alpha_1} s_1 \xrightarrow{[\phi_2]\,\alpha_2} s_2$, with α_1 and α_2 do not satisfy the condition above, we introduce an additional transition at s_1, *e.g.*, $s \xrightarrow{[\phi_1]\,\alpha_1} s' \xrightarrow{\alpha} s_1 \xrightarrow{[\phi_2]\,\alpha_2} s_2$. The event α does not carry a variable but it connects α_1 and α_2. Sender and receiver of α is determined based on communication modes. If the sending mode is selected, then the sender of α is either the sender or the receiver of α_1, while the receiver of α is either the sender α_2. Thus with the additional interaction, the connectedness between α_1 and α, then between α and α_2 is guaranteed. For example, to ensure the connectedness of `ship` and `confirm` in our running example, with the sending mode, an additional interaction from the shipper to the buyer is added as follows: $s \xrightarrow{ship^{[w,s]}.x_0} s' \xrightarrow{additional^{[w,v]}} s_1 \xrightarrow{confirm^{[v,b]}.\langle x_3\rangle} s_2$. Let us remark that, the interaction $additional^{[s,v]}$ may be also added.

B. Data connectedness

In a choreography, a variable indicates not necessarily explicitly its owned roles. However, in implementation, to validate a guard, a role must know the free variables that appear in the guard. Hence any free variables used by a role (in guards or in free interactions) must be known by itself. Intuitively, a variable is known when its value is pre-configured before running time or its value is received from another role. Let us remark that, value of variable x can be changed after a bound interaction $o^{[a,b]}.\langle x\rangle$ occurs, hence only a and b know x when this interaction happens, other roles could know x if there exists some free interactions carrying x from either a or b to them.

If a role c uses a variable x in its guards or in its free interactions, but c does not know x, *e.g.*, $s \xrightarrow{o_1{}^{[a,b]}.\langle x\rangle} s_1 \xrightarrow{[x>0]\; o_2{}^{[c,d]}.x} s_2$, then an additional interaction will be introduced in order to connect data of variable x from the roles knowing it, a or b, to c, *e.g.*, $s \xrightarrow{o_1{}^{[a,b]}.\langle x\rangle} s' \xrightarrow{additional^{[b,c]}.x} s_1 \xrightarrow{[x>0]\; o_2{}^{[c,d]}.x} s_2$. The proposition of additional interactions in order to connect data is done as in Algorithm 1. In which a proposed additional interaction is in the form $\langle t, \{a,b\}, c, x\rangle$ where t is the transition which uses variable x, hence additional interaction will be inserted before it, and $\{a,b\}$ is a set of roles which knows x and will send x to c.

Algorithm 1: Correction of data connectedness

Input: a global STG $\mathcal{C} = (S, s_0, T)$
Output: set of additional interactions
let $owner(s,x)$ as set of roles which know variable x before state s;
$E := \emptyset$; // set of additional interactions
foreach transition $s_i \xrightarrow{[\phi_i]\,\alpha_i} s_i'$ **in** T **do**
  $V := \mathtt{fv}(\phi_i) \cup \mathtt{fv}(\alpha_i)$; // set variable to be verified
  $a := \mathtt{sender}(\alpha_i)$;
  foreach $v \in V$ **do**
    if $(a \notin owner(s_i, v))$ **then** add $\langle t, owner(s_i, v), a, v\rangle$ to E;
  // update *owner*
  if (α_i *is a bound interaction* $o^{[a,b]}.\langle x\rangle$) **then**
  $owner(s_i', x) := \{a, b\}$;
  if (α_i *is a free interaction* $o^{[a,b]}.x$) **then**
  $owner(s_i', x) := owner(s_i, x) \cup \{b\}$;
return E;

C. Branching decision

A state s of a STG may have many outgoing transitions. Each transition has a guard. If these guards are not implied each others, then the branching at the state s is firstly decided by these guards. Obviously, this condition cannot be determined at syntax level but at semantic level. In such a case, s is a *conditional branching*. Thanks to these guards, only one branch can be selected at a time, and each initial role can always compute whenever it does or does not initiate the interaction.

Contrarily, if the state s is an *unconditional branching*, there are several guards that may be satisfied, hence many transitions

may be fired. The branching will be decided by the roles which initiate interactions of each outgoing transition. A transition $s \xrightarrow{[\phi]\,\alpha} s'$ may be initiated by either sender or receiver of α in synchronous mode, but only by the sender of α in asynchronous mode. State (2) of the STG in Figure 4(b) is an unconditional branching, while the state (7) is a conditional branching.

When many roles may decide the route, but these roles works independently, the choreography will not be respected since each role may select a different branch. Therefore, only one initiator is allowed to decide the route. If a branching has more than one initiator, a dominant initiator will be selected to decide the route. Dominant initiator is the role which may initiate maximal transitions, *e.g.*, at the state (2), $buyer$ is a dominant role. Some additional interactions from the initiator to the other ones will be introduced in order to complete the control of the dominant initiator into these branches. Hence the selection of dominant initiator adds minimal interactions. Some additional interactions from the initiator to the other roles may be also introduced to inform them of the selected route.

As the natural projection, see Table II, guards are preserved only at local model of the sender. However, in the case of conditional branching, guards are also maintained at local models which decide the route based on these guards. A conditional branching can be also decided as done for an unconditional branching. However, the conditional branching (based on data) is better than unconditional one (based on events) since less additional interactions are added. Let us consider a state s having m branches, of a choreography STG with n roles. If s is a conditional branching, then we need no more than n additional interactions which correspond to the n roles involved in the branching. Otherwise, if s is an unconditional branching, there are no more than $(n-1) \times m$ additional interactions from dominant role to $n-1$ others in order to inform selected branches. Most of existing work does not deal with the case of conditional branchings. The correction of branching is done as depicted by the pseudo-code in Algorithm 2.

Algorithm 2: Correction of branching

```
   Input: a choreography C = (S, s_0, T)
   Output: set of additional interaction
1  let R as set of roles in C;
2  E := ∅ ;                          // set of additional interactions
3  foreach s ∈ S do
4      if (|outgoing(s)| = 1) then continue ;        // no branching
5      isConditional := true;
6      foreach transition s --[φ1]α1--> s1, s --[φ2]α2--> s2 in outgoing(s) do
7          // whether exists some value of variables s.t.
               φ1 and φ2 are satisfied
8          if (SOLVE(φ1 ∧ φ2)) then
9              isConditional := false;          break;
10     if (isConditional) then continue ; // conditional branching
11     // branching deciding by initiators
12     select a dominant role a that participates the most of outgoing transition;
13     foreach transition t_i := s --[φi]αi--> s_i in outgoing(s) do
14         b := sender(α_i);          c := receiver(α_i);
15         foreach r ∈ R do
16             if (a ≠ r) ∨ (a ≠ b ∧ r ≠ c) then add ⟨t, {a}, r⟩ to E;
17 return E;
```

The output of the projection of the global STG, described in Figure 4(b) under synchronous communication mode, produces four local STGs, as shown in Figure 6, corresponding to local models of the buyer (a), the vendor (b), the warehouse (c), and the shipper (d). For the sake of clarity, in these local STGs, the additional states are gray and the additional interactions start with $+$, *e.g.*, $+br_i$ for selecting branch i, $+cbr$ for connecting data, and $+or$ for ordering events.

VI. Experimental Evaluation

The approach proposed in this paper is fully tool–supported. The tool is freely available online[1]. There exists also a web-based application version of the tool, hence one may try it directly with a web browser without any required configurations. It takes as input a script file which contains global STGs representing the choreographies to be analyzed. The output is a set of local STGs each one representing the required behaviors of each role in each choreography.

A. Boolean Condition Solver

The unreachable transition and conditional branching, *e.g.*, the function $\texttt{SOLVE}(\phi)$ in the above algorithm, are checked with the aid of the Z3 SMT solver[2]. Let us consider an example of checking whether the branch at state 7 in Figure 4(b) is a conditional branching. We need to check whether there exists some values of x_2 such that both the first branching condition $(x_2 = \texttt{OK})$ and the second branching condition $(x_2 \neq \texttt{OK})$ are *true*. The translation in a Z3 script to check this example is as follows:

```
1  (declare-sort A)
2  (declare-const OK A)
3  (declare-fun x2 () A)
4  (define-fun phi1 () Bool (= x2 OK))
5  (define-fun phi2 () Bool (not (= x2 OK)))
6  (assert (and phi1 phi2))
7  (check-sat)
```

Specially, we first declared a data sort `A` as a general data type. Then `OK` (resp. x_2) is declared as a constant (resp. variable) typed `A`. Two boolean functions ϕ_1 and ϕ_2 are defined based on relation between x_2 and `OK`. This very simple example shows that these functions, variable and constant are uninterpreted, *i.e.*, Z3 does not use their interpretation (concrete values) but relies on their definition supported by a dedicated decision procedure. The `check-sat` command checks if equation $(\phi_1 \wedge \phi_2)$ defined by the `assert` command is *true* for some interpretations of its variables. If this is the case, the response of `check-sat` is `sat`, otherwise, `unsat`, *i.e.*, the equation is always *false*. In the example, the response will be *unsat*.

B. Experiments

The experimental evaluation of the algorithm for realizability checking and choreography projection in this section was performed on a desktop computer running 32-bit XUbuntu 13.4 (kernel 3.8.0-23-generic) with Intel Pentium 4 3.2GHz processors and 1GB of RAM.

We first applied our tool to our choreography running example (OSP). We then demonstrate the scalability of the tool by applying it on examples from the literature as illustrated in

[2]http://research.microsoft.com/en-us/um/redmond/projects/z3/

Table III. The rows of the table correspond to the choreographies: Request For Quota (RFQ [19]), Train Station Services (TSS [20]), Market [21], and our running example. The second column corresponds to the inputs which are choreography specifications defined by the number of: transitions (#Trans.), states (#States), roles (#Roles) and operations (#Ops.). The remaining columns are devoted to represent the checking result of: the reachability with their verdicts (Vdict) and number of cut-off interactions (#delInt.); the realizability, with their verdicts (Vdict) and number of additional interactions (#addInt.), corresponding to the four cases of communication modes, synchronous (sync.), sending, reception, and disjoint. The last column gives the time needed for the checking. To verify our projection, on one hand, we recomposed the generated local models to obtain one STG, representing the composition, which was then compared against the choreography STG. On the other hand, since local models of the examples were available, we compared them with our generated local models. The comparisons were done thanks to our Symbolic Branching Bisimulation Conformance tool [9] in which the conformation relation between two STG models is based on symbolic branching bisimulation [9], [12]. The experiments showed that our projections were correct. The experiments can be reproduced on the web site[1]. The composition is done with the operator $||$ and the conformance checking with the command `conformance`.

Additional interactions are generated to solve the conflicts related to choreography order interaction, to branches selection and to variables values. The Figure 7 illustrates how the number of additional interactions is related to the number of states, branches for each state, roles, operations and data for the choreography under the assumption of synchronous communication mode. To conduct the experiments, we started from an initial configuration defined by 5 parameters and their values, *i.e.*, 1000 states, 5 branches, 5 roles, 20 operations, 1 data level. With this configuration we generated a choreography STG with a tree structure and it is a 5-ary tree. Data level 1 denotes that variables are carried by events for all the transitions of the choreography. Data level $n > 1$ means that variables are still carried by events and are also on guards. Consequently, the data level increases with the number of variables in the guard. Based on the initial configuration, we produced several other configurations by only varying one parameter. For instance, to test the impact of the number of states, as illustrated in Figure 7(a), we used 10 configurations: (states: x, branches: 5, roles: 5, operations: 20, data: 1) with $x \in \{1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000\}$. Based on each configuration, we generated randomly 30 choreography STGs. We then tested on each generated STG. The result of one tested configuration is the average of the results of these 30 STGs.

In our formal model, only one variable is changed when an interaction happens. Hence if a role on the next transition depends on this variable, we need to introduce only one additional transition to transfer this variable to the concerned role. To ensure the ordering of interactions, additional events are added. Indeed, when an interaction e occurs, m interactions may happen after e, *e.g.*, the target state of e may have m outgoing transitions. In such a case, there are no more than m additional interactions to add in order to solve the order conflict between e and m interactions. For the branching decision, as explained in Section V-C, there are no more than $(n-1) \times m$ additional interactions from a dominant role. This latter has to inform $n-1$ roles of the selected branches. The number of additional interactions for each configuration consists of three parts, for data, branching and ordering. Generally, the proportions of theses parts, data:ordering:branching, are $1{:}m{:}m \times (n-1)$.

TABLE III. EXPERIMENTAL RESULTS

Name [Ref.]	Choreography Size (#Trans./#States/#Roles/#Ops.)	Reachability (Vdict./#delInt.)	Realizability (Vdict./#addInt.)				Duration (seconds)
			sync.	sending	reception	disjoint	
RFQ [19]	8/8/6/3	yes/0	yes/0	yes/0	no/1	no/1	0.041
TSS [20]	12/11/4/9	yes/0	yes/0	no/2	no/1	no/4	0.125
Market [21]	10/10/4/9	yes/0	yes/0	yes/0	no/1	no/1	0.077
OSP [ours]	11/9/4/9	yes/0	no/5	no/7	no/7	no/7	0.060

To confirm our analysis on the formal model, we have conducted several experiments that are depicted in Figure 7. The mentioned proportions are confirmed by the obtained results for each bar of the bar chart. Furthermore, we can exhibit with these experiments several features of our projection. In Figure 7(a), we observe that the number of interactions increases with the number of states. In this case, the tree becomes deeper. In Figure 7(b), when we vary the number of branches, we obtain a larger tree with a higher number of leaves, as a consequence the number of intermediate states will decrease. This decrease will slow down the increase of the number of additional interactions used for branching decisions. However the number of intermediate states decreases slowly when number of branches reaches the value 25 and higher values. That is why we observe that the number of additional interactions reaches a threshold. Let us note that for each experiment only one parameter was changed and especially here the number of states was not changed, it is why we obtain a larger tree. Consequently, for projection concerns, it is notably better to construct a choreography in breadth rather in depth. Figure 7(c) shows that our method is more efficient with a higher number of roles. Indeed, in this case, for each state, we determine a dominant role which will have in charge to inform the others of the selected route. However the dominant role does not need to inform a role which is a receiver of an interaction e on a branch since it is informed by the sender of e. Consequently, the dominant role needs to inform less roles when the number of roles (act as receivers) increases. Hence this slows down the increase of the number of additional interactions by increasing the number of roles. In Figure 7(d), we increase the number of operations, in others words the alphabet. Such a change has no impact on the choreography realizability as the complexity of the choreography (initial configuration) is not modified. Indeed, only the label, *i.e.*, operations name, are changed. In Figure 7(e) the number of variables is changed. We observe that when we increase the number of variables, the number of additional interactions increases in order to ensure the data-connectedness (the top bar is increasing). To conclude, hopefully the complexity of the projection with value-passing is not totally dependent on the data complexity. The verification times are shown in Figure 7(f). The horizontal axis represents the 10 configurations. It illustrates once again the dependability of realizability on the complexity of choreography.

VII. RELATED WORK

In this section, we put our work in context of the existing choreography modeling and realizability analysis approaches

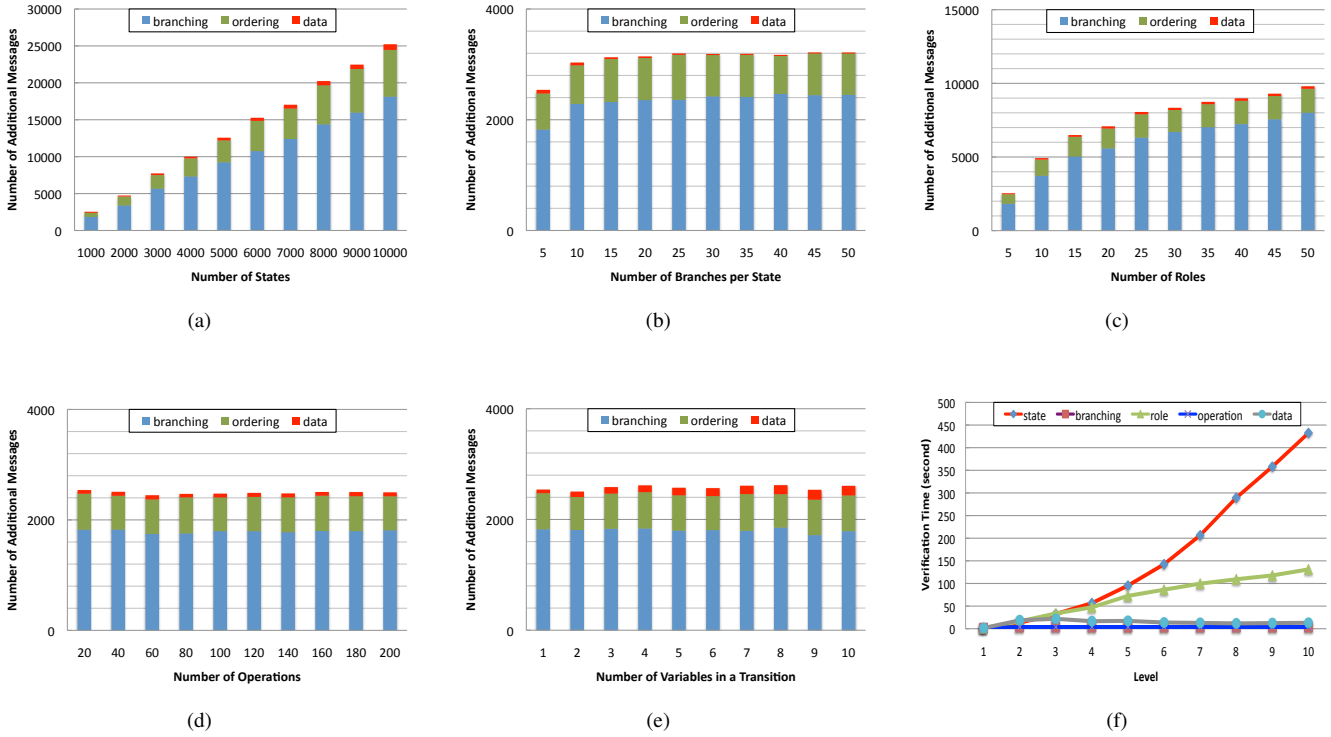

Fig. 7. Impact of number of (a) states, (b) branches, (c) roles, (d) operations and (e) variables on the realizability of choreography; and (f) verification time

for service choreographies. The comparison is presented in Table IV. Such a comparison could not be done from a tool perspective (in the previous section) since most of the related work is not tool-equipped.

A. Data-Aware Interaction-based Choreography Models

Columns 2, 3 and 4 are relative to the way data is supported. In [19], the authors discuss about using symbolic model checking to analyze choreography but its choreography model still bounds value domain of variables and no tool is available.

Columns 5 and 6 present the expressiveness of choreography model. Having both loops and assignments may yield state space explosion if one does not close the system or bound data domains. [26]–[28] avoids state space explosion when checking the realizability based on a syntactic analysis of choreography specification. This will miss the cases of decision based on data and unreachable transitions/events as discussed in Section II. In this work, we do support loops and a limited form of assignments through bound events.

TABLE IV. SERVICE CHOREOGRAPHY MODELING APPROACHES

	Data			Expressiveness		Projection	
	support	data-aware interaction	treatment	loops	assign.	basic	smart
[22]	no			yes	no	yes	partial
[23]				yes	no	yes	no
[20]				yes	no	yes	yes
[24]				yes	no	yes	yes
[25]	yes	no	closure	yes	yes	no	no
[26]				yes	yes	yes	no
[21]				no	yes	no	no
[27]				yes	yes	yes	no
[28]				no	yes	yes	yes
[19]			bound data	yes	no	yes	no
[2]		yes	bound data	no	no	yes	no
ours			symbolic	yes	limited	yes	yes

B. Realizability Checking & Endpoint Projection

Two last columns focus on the endpoint projection. In a top-down service choreography approach, service developers develop a global specification and project it into local specifications. Services are then selected to implement the local specifications. Existing approaches render the realizability of choreography by introducing extra interactions [20], [22], [24], [28]. Some other approaches enable the realizability by privileging some roles in the choreography, *e.g.*, dominant role in [22] which decides the route in the case of non-deterministic branching, or additional coordinators in [24], [29]. However, the latter approaches seem to be inappropriate in the sense of the choreography where there is no privileged roles. Furthermore these approaches do not say how to ensure the connectedness of events and they do not support data.

The work closes to ours is presented in [2], [28]. In [28], the authors propose a way to repair choreography to make it realizable and then apply the projection on it. However, this work deals with ground values and it is done by analyzing the syntax of choreography specification. In [2], the choreography model is formalized by using an extension of Petri–Net where value domain of variables is also bounded. The authors focus on modeling choreography by extending BPMN 2.0 choreography to support value-passing rather than endpoint projection. We use this extending to describe our OSP running example as in Figure 4(b). The realizability is analyzed at local traces which are projected from the traces of choreography. The work does not propose any solution to ensure the projection (global conformance) and to enable the realizability.

The choreography modeling and realizability checking are also presented in [30]–[33]. The realizability is checked in [30] based on access control policies of local services and the credentials that local services are willing to disclose. [31], [32] propose tools for checking the realizability of choreography modeled by collaboration diagrams and by BPMN 2.0. The necessary and sufficient conditions for realizability of choreographies are given in [33]. However, all these approaches do not explicitly support data exchanged by messages and used for routing decisions.

VIII. Conclusion and Future Work

Supporting data in service choreography modeling and analysis is a crucial issue. To address this issue, we first propose a symbolic interaction-based choreography model with data-aware interaction as the basic event. This model enables us to analyse choreographies in presence of data without suffering from state space explosion and without requiring to bound data domains. Going further than checking realizability, we propose a projection to generate behavioral peer skeletons with the minimum extra interactions that ensure conformance to the choreography. The projection supports both synchronous and asynchronous communication modes. We have implemented our techniques for realizability checking and projection. Our tool is freely available online[1]. It includes a data-aware choreography language that we have defined, and a transformation from this language to our choreography model.

Future work concerns the definition of model transformations from standard choreography languages, namely WS-CDL and BPMN 2.0. Furthermore, we will study the distributed testing of choreography implementations based on the generation of test cases or properties from the peer skeletons.

Acknowledgment

This work is supported by the Personal Information Management through Internet project (PIMI-ANR-2010-VERS-0014-03) of the French National Agency for Research.

References

[1] G. Decker, O. Kopp, and A. Barros, "An Introduction to Service Choreographies," *Information Technology*, vol. 50, no. 2, pp. 122–127, 2008.

[2] D. Knuplesch, R. Pryss, and M. Reichert, "Data-aware Interaction in Distributed and Collaborative Workflows: Modeling, Semantics, Correctness," in *Proc. of CollaborateCom'12*, 2012.

[3] S. Pavel, J. Noyé, P. Poizat, and J.-C. Royer, "A Java Implementation of a Component Model with Explicit Symbolic Protocols," in *Proc. of SC'05*, 2005.

[4] F. Fernandes and J.-C. Royer, "The STSLib Project : Towards a Formal Component Model Based on STS," in *Proc. of FACS'07*, 2007.

[5] F. Fernandes, R. Passama, and J.-C. Royer, "Components with Symbolic Transition Systems: a Java Implementation of Rendezvous," in *Proc. of CAP'07*, 2007.

[6] L. Bentakouk, P. Poizat, and F. Zaïdi, "A Formal Framework for Service Orchestration Testing based on Symbolic Transition Systems," in *Proc. of TESTCOM'09*, 2009.

[7] R. Mateescu, P. Poizat, and G. Salaün, "Adaptation of service protocols using process algebra and on-the-fly reduction techniques," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 755–777, 2012.

[8] J. Cámara, G. Salaün, C. Canal, and M. Ouederni, "Interactive Specification and Verification of Behavioral Adaptation Contracts," *Information & Software Technology*, vol. 54, no. 7, pp. 701–723, 2012.

[9] H. N. Nguyen, P. Poizat, and F. Zaïdi, "A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies," in *Proc. of ICSOC'2012*, 2012.

[10] O. Kopp and F. Leymann, "Do We Need Internal Behavior in Choreography Models?" in *Proc. of ZEUS'09*, 2009.

[11] M. Hennessy and H. Lin, "Symbolic bisimulations," *Theoretical Computer Science*, vol. 138, no. 2, 1995.

[12] R. Van Glabbeek and W. Weijland, "Branching Time and Abstraction in Bisimulation Semantics," *Journal of the ACM*, vol. 43, no. 3, pp. 555–600, 1996.

[13] W. Deng and H. Lin, "Extended Symbolic Transition Graphs with Assignment," *Proc. of COMPSAC'05*, 2005.

[14] J. Pathak, S. Basu, R. Lutz, and V. Honavar, "MoSCoE: An Approach for Composing Web Services through Interative Reformulation of Functional Specification," *International Journal on Artificial Intelligence Tools*, vol. 17, no. 1, pp. 109–138, 2008.

[15] C. Gaston, P. L. Gall, and N. Rapin, "Symbolic Execution Techniques for Test Purpose Definition," in *Proc. of TESTCOM'06*, 2006.

[16] H. N. Nguyen, P. Poizat, and F. Zaïdi, "Online Verification of Value-Passing Choreographies through Property-Oriented Passive Testing," in *Proc. of HASE'2012*, 2012.

[17] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro, "Bridging the Gap between Interaction- and Process-Oriented Choreographies," in *Proc. of SEFM'08*, 2008.

[18] H. N. Nguyen, P. Poizat, and F. Zaïdi, "Passive Conformance Testing of Service Choreographies," in *Proc. of SAC'12*, 2012.

[19] R. Kazhamiakin and M. Pistore, "Choreography Conformance Analysis : Asynchronous Communications and Information Alignment," in *Proc. of WS-FM'06*, 2006.

[20] G. Salaün, T. Bultan, and N. Roohi, "Realizability of Choreographies Using Process Algebra Encodings," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 290–304, 2012.

[21] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, "Choreography and Orchestration Conformance for System Design," in *Proc. of COORDINATION'06*, 2006.

[22] Z. Qiu, X. Zhao, C. Cai, and H. Yang, "Towards The Theoretical Foundation of Choreography," in *Proc. of WWW '07*, 2007.

[23] M. Bravetti and G. Zavattaro, "Towards a Unifying Theory for Choreography Conformance and Contract Compliance," in *Proc. of SC'07*, 2007.

[24] G. Diaz and I. Rodriguez, "Automatically Deriving Choreography-Conforming Systems of Services," in *Proc. of SCC'09*, 2009.

[25] H. Yang, X. Zhao, Z. Qiu, G. Pu, and S. Wang, "A Formal Model for Web Service Choreography Description Language (WS-CDL)," in *Proc. of ICWS'06*, 2006.

[26] J. Li, H. Zhu, and G. Pu, "Conformance Validation between Choreography and Orchestration," in *Proc. of TASE'07*, 2007.

[27] Z. Xiangpeng, Y. Hongli, and Q. Zongyan, "Towards the Formal Model and Verification of Web Service Choreography Description Language," in *Proc. of WS-FM'06*, 2006.

[28] J. Sun, Y. Liu, J. S. Dong, G. Pu, and T. H. Tan, "Model-Based Methods for Linking Web Service Choreography and Orchestration," in *Proc. of APSEC'10*, 2010.

[29] M. Autili, D. Di Ruscio, A. Di Salle, P. Inverardi, and M. Tivoli, "A Model-Based Synthesis Process for Choreography Realizability Enforcement," in *Proc. of FASE'13*, 2013.

[30] F. Paci, M. Ouzzani, and M. Mecella, "Verification of access control requirements in web services choreography," in *Proc. of SCC'08*, 2008.

[31] T. Bultan, C. Ferguson, and X. Fu, "A Tool for Choreography Analysis Using Collaboration Diagrams," in *Proc. of ICWS'09*, 2009.

[32] P. Poizat and G. Salaün, "Checking the Realizability of BPMN 2.0 Choreographies," in *Proc. of SAC'12*, 2012.

[33] S. Basu, T. Bultan, and M. Ouederni, "Deciding Choreography Realizability," in *Proc. of POPL'12*, 2012.