A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies*

Huu Nghia Nguyen[1], Pascal Poizat[1,2], Fatiha Zaïdi[1]

[1] LRI; Univ. Paris-Sud, CNRS, Orsay, France
[2] Univ. Évry Val d'Essonne, Evry, France
{huu-nghia.nguyen, pascal.poizat, fatiha.zaidi}@lri.fr

Abstract. Checking choreography conformance aims at verifying whether a set of distributed peers or local role specifications match a global specification. This activity is central in both top-down and bottom-up development processes for distributed systems. Such systems usually collaborate through information exchange, thus requiring value-passing choreography languages and models. However, most of the conformance checking techniques abstract value-passing or bound the domains for the exchanged data. As an alternative, we propose to rely on symbolic models and an extension of the symbolic bisimulation equivalence. This enables one to take into account value passing while avoiding state space explosion issues. Our framework is fully tool supported.

Keywords: choreography, specification, conformance, symbolic transition systems, symbolic branching bisimulation, tools.

1 Introduction

Context and Issues. A *choreography* is the description with a global perspective of *interactions* between *roles* played by *peers* (services, organizations, humans) in some collaboration. One key issue in choreography-based development is checking the *conformance* of a set of local descriptions wrt. the choreography global specification. This issue naturally arises both in bottom-up and in top-down development processes [1], and is also a cornerstone for realizability checking. The definition of conformance should not be too strict. It should support *choreography refinement*, *e.g.*, with peers and interactions being added in the implementation by the service architect in order to enforce the specification. Finally, entities in a distributed system usually exchange information, *i.e.*, data, while interacting. Consequently, *data should be supported* in choreography specifications, in the descriptions of the local entities, and in the conformance relation.

Related Work. In Table 1, we compare related conformance checking approaches. Columns 2 and 3 focus on data support. Some approaches [2–4] *abstract data away*. This is known to yield over-approximation issues, *e.g.*, false negatives in the

* This work is supported by the PIMI project (ANR-2010-VERS-0014-03) of the French National Agency for Research.

Table 1. Choreography conformance approaches

	Data & Value-Passing		Expressiveness		Conformance	
	supported	treatment	loops	assignment	global	relation (based on)
[2]	no	-	yes	no	yes	Trace equivalence
[3]			yes	no	yes	Weak bisimulation
[4]			yes	no	yes	Strong bisimulation
[5]	yes	closure	yes	yes	no	Weak bisimulation
[6]		closure	no	yes	yes	Branching bisimulation
[7]		bound data	yes	no	yes	Branching bisimulation
this paper		symbolic	yes	limited	yes	Branching bisimulation

verification process. Data can be supported by working on *closed implementation-level systems* where sent messages contain only ground data [5, 6]. In such a case, the state space explosion of the system model is limited. However, this is not adequate when working on abstract specifications where there are no such ground sent messages but only free variables and constraints on their values. Another solution is to *bound data domains*. The issue is that conformance may not yield outside the bounds. Defining bounds in order to avoid false positives in the verification process can be difficult. In our framework, data is supported using a *symbolic approach* and conformance may be checked for whole data domains.

Columns 4 and 5 are relative to choreography *expressiveness*. Having both loops and assignments may yield state space explosion if one does not close the system or bound data domains. In this work, we do support loops and a limited form of assignment through message reception.

The last two columns are relative to the kind of conformance being supported and the behavioural equivalence being used. *Global conformance* is important in conformance checking since one wants not only to know if each peer is conform to its role, *i.e.*, *local conformance*, but also if the peers altogether have a behaviour that is conform to the choreography. Local conformance does not implies global conformance. Weak and branching bisimulations are able to support internal actions and hiding (formally, τ actions). This is important, *e.g.*, if one has to deal with messages added to make some choreography realizable. Branching bisimulation [8] has been preferred over weak bisimulation in the last years since it is a congruence, hence supports compositional reasoning.

Symbolic bisimulations, defined on Symbolic Transition Graphs (STGs), have been introduced in [9] with both early and late semantics. In this work, we use a late semantics. STGs have then been extended to STGs with assignments (STGAs) in [10, 11]. These works mostly concentrate on strong and weak bisimulation. Symbolic branching bisimulation has not yet received much attention. As a consequence, there is tool support for symbolic strong bisimulation [12] but not for symbolic branching bisimulation.

Contributions. Our contributions are the following. Based on process algebras for choreography [2, 13], we propose *a specification and description language* addressing both the *global* (choreography) and the *local* (peers description, role requirements) perspective over distributed systems. Our language supports *information exchange and data-related constructs* (conditional and loop constructs).

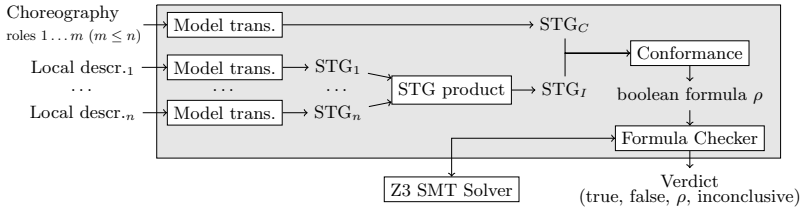


Fig. 1. Architecture of our framework

We give a *fully symbolic semantics* to this language using a model transformation into STGs, thus avoiding data abstraction and over-approximation, restriction to manually bound data domains, and limitation to implementation-level closed descriptions. Accordingly, we build on branching bisimulation [8] and on a symbolic extension of weak bisimulation [11] to develop a specific *symbolic version of branching bisimulation* dedicated at *checking the conformance* of a set of local entities wrt. a choreography specification. Our equivalence enables one to check conformance in presence of choreography *refinement*, *i.e.*, where new peers and/or interactions may be added wrt. the specification. Going further than a *true* vs. *false* result for conformance, our approach supports the generation of the *most general constraint over exchanged information* in order to have conformance. Finally, our framework is *fully tool supported*[3].

In the sequel, we present the principles of our approach (technical detail can be found in [14]) and we end with conclusions and perspectives of our work.

2 Architecture of the Framework

In this section, we introduce our framework for choreography conformance checking. We also present some of the experiments we have made to evaluate it. The architecture of our framework is given in Figure 1. We take as input a choreography global specification C, with m roles. We also take an implementation description I, given as $n \geq m$ entity local descriptions. These may correspond either to peer descriptions or to role requirements. The case when $n > m$ denotes, *e.g.*, an implementation where some peers have been added to make a choreography realizable. All inputs are first transformed into STGs. The product of STGs and the restriction to actions in C are used to retrieve a unique STG for I, thus yielding two STGs to compare: one for C ($\mathcal{C}$) and one for I ($\mathcal{I}$). We then check if $\mathcal{I}$ conforms to $\mathcal{C}$, which generates the largest boolean formula ρ such that the initial states of $\mathcal{I}$ and $\mathcal{C}$ are conformance related. Finally, this formula is analysed using the Z3 SMT solver[4] in order to reach a conformance verdict. This can be "always true" or "always false", "always" meaning whatever the data values exchanged between peers are. However, sometimes we can have conformance only for a subset of these values. Going further than pure true/false

[3] Our tool is freely available at http://www.lri.fr/~nhnghia/sbbc/
[4] http://research.microsoft.com/en-us/um/redmond/projects/z3/

conformance, our framework thus allows to compute the largest constraint on data values, ρ, that would yield conformance. Complex constraints may cause the solver to return a timeout. In such a case, we emit inconclusiveness as a verdict.

A Language for Choreographies, Roles, and Peers. Since we are interested in an abstract, *i.e.*, implementation independent, formal choreography language, we choose an interaction-based model [15] and the usual τ actions can be ignored [16]. Our specification language, inspired by [2, 13], is used to specify distributed systems with a global perspective, *i.e.*, *choreographies*, to define local requirements, *i.e.*, *roles*, and to describe the pieces of a distributed implementation, *i.e.*, *peers*. Due to this multi-purpose objective, it is first presented in terms of an abstract alphabet, A. We then explain how A can be realized for the different purposes. The syntax of our specification language, $L(A)$, is given by:

$$L ::= \mathbf{1} \mid \alpha \mid L; L \mid L + L \mid L|L \mid L[>L \mid [\phi] \triangleright L \mid [\phi] * L$$

A basic activity is either termination ($\mathbf{1}$) or a regular activity $\alpha \in A$. Structuring is achieved using sequencing ($;$), non deterministic choice ($+$), parallelism ($|$), and interruption ($[>$). Furthermore, we have data-based conditional constructs, namely guards ($\triangleright$) and loops ($*$), where ϕ is a boolean expression.

The basis of an interaction-model choreography description is the interaction. Let us denote an interaction c from role a to role b by $c^{[a,b]}.x$, where x is a variable that represents the information exchanged during interaction (x is omitted when there is none). We stress out that x can be structured, *e.g.*, to denote a multiple data exchange as done in Web services with XML message types. A *choreography specification* for a set of roles R, a set of interactions Ch, and a set of variables V, is an element of $L(A)$ with $A = \{c^{[a,b]}.x \mid c \in Ch \wedge a \in R \wedge b \in R \wedge a \neq b \wedge x \in V\}$.

The events of a local entity a (peer or role) can be abstracted as sending and reception events, denoted respectively with $c^{[a,b]}!x$ and $c^{[b,a]}?x$, where b is another entity, *i.e.*, $a \neq b$, and x is the information exchanged during interaction (x is omitted when there is none). An *entity description* for an entity a, a set of roles R with $a \in R$, a set of interactions Ch, and a set of variables V, is an element of $L(A)$ with $A = \{c^{[a,b]}!x,\ c^{[b,a]}?x \mid c \in Ch \wedge b \in R \wedge a \neq b \wedge x \in V\}$.

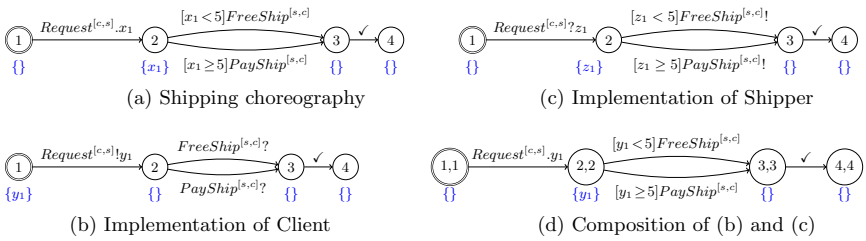
Example 1. Let us suppose a shipping choreography between two roles: c (client) and s (shipper). The client first requests shipping by providing the weight of goods to be sent. If this is less than 5 kgs then the goods will be sent for free. Otherwise, the shipping has to be paid. This can be described as follows:

$$\mathit{Shipping} ::= \mathbf{Request}^{[c,s]}.x_1; ([x_1 < 5] \triangleright \mathbf{FreeShip}^{[s,c]} + [x_1 \geq 5] \triangleright \mathbf{PayShip}^{[s,c]})$$

A tentative to implement the shipping choreography is that the client sends the weight to the shipper and then waits for either free or paid shipping, while it is the shipper that checks the weight in order to decide which shipping is used:

$$\mathit{Client}\ c ::= \mathbf{Request}^{[c,s]}!y_1; (\mathbf{FreeShip}^{[s,c]}? + \mathbf{PayShip}^{[s,c]}?)$$
$$\mathit{Shipper}\ s ::= \mathbf{Request}^{[c,s]}?z_1; ([z_1 < 5] \triangleright \mathbf{FreeShip}^{[s,c]}! + [z_1 \geq 5] \triangleright \mathbf{PayShip}^{[s,c]}!)$$

Symbolic Transition Graph. An STG [9] is a transition system where a set of variables, possibly empty, is associated to each state and where each transition may be guarded by a boolean expression ϕ that determines if the transition can



(a) Shipping choreography

(c) Implementation of Shipper

(b) Implementation of Client

(d) Composition of (b) and (c)

Fig. 2. STGs for Example 1

be fired or not. Actions labelling transitions will correspond in our work to the elements of the alphabets we have seen earlier on. We also add a specific event, ✓, to denote activity termination. A transition from state s to s' with a guard ϕ and labeled by an action α takes the form $s \xrightarrow{[\phi]\ \alpha} s'$. We use STGs as a formal model to give semantics to our language. The product of STGs is used to give a semantics to a set of interacting local entities. We assume that the STGs use disjoint sets of variables which can be achieved using, *e.g.*, indexing by the name of the entity. The rule-based of model transformations and our algorithm for the product of STGs can be found in a technical report extension of this paper [14].

Example 2. The STGs for the choreography, the client and shipper in Example 1 are shown in Figure 2(a-c). Figure 2(d) presents the product of the STGs in Figure 2(b) and Figure 2(c). The free variables of the states are given below them, *e.g.*, $\{x_1\}$ for state 2 in the choreography STG.

Choreography Conformance. Since our semantic models are STGs, we define conformance over two STGs, $\mathcal{I}$ (implementation) and $\mathcal{C}$ (choreography). We choose branching bisimulation [8] as a basis since it supports equivalence in presence of τ actions that result from the hiding of interactions added in implementations wrt. specifications, *i.e.*, refinement. However, branching bisimulation is defined over ground terms (no variables), while STGs may contain free variables. In [6, 7], this issue is considered by introducing at each state an evaluation function that maps variables to values, thus reducing open terms to ground ones. This may lead to state space explosion when domains of the variables are big. Alternatively, we base our work on (late) symbolic extensions of bisimulations, introduced in [9–11], that directly support open terms.

To make implementation and specification comparable, we remind the reader that we assume the two STGs have disjoint sets of variables which can be achieved using, *e.g.*, indexing. We also assume that a local entity has the same identifier than the corresponding role in the choreography. This constraint could be lifted using a mapping function. Additional interactions may have been introduced in the implementation wrt. the specification during refinement, *e.g.*, to make it realizable. In order to compare the STGs, we have to hide these interactions.

Example 3. We give a refinement example in Figure 3(a) where the client specifies the maximum (s)he agrees to pay for the shipping (y). This influences the sequel

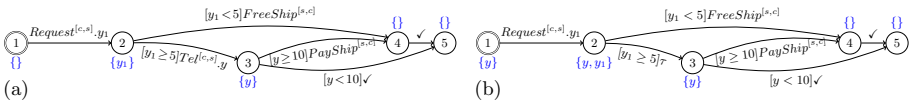


Fig. 3. A refinement (a) for the STG in Figure 2(d) and its restriction (b)

of the implementation since the non-free shipping costs \$10: if the user requires to pay less, no shipping is done. The restriction of this STG to the set of actions used in the choreography specification, $\{Request^{[c,s]}, FreeShip^{[s,c]}, PayShip^{[s,c]}\}$, yields the STG in Figure 3(b) where *Tel* has been hidden.

Conformance computation. Our algorithm for the computation of the conformance relation between two STGs [14] is a modification and simplification of the one proposed in [11] that computes symbolic weak bisimulation. Simplification was made possible since there may be τs in $\mathcal{I}$ (after hiding) but not in $\mathcal{C}$. The algorithm outputs a set of boolean formulas ρ_{s_1,s_2} relative to pairs of states (s_1, s_2), s_1 being in $\mathcal{I}$ and s_2 in $\mathcal{C}$. ρ_{s_1,s_2} denotes the conditions under which s_1 and s_2 are conformance related. In the algorithm, these boolean formulas are encoded as a Predicate Equation Systems (PES) [10], *i.e.*, a set of functions each of which contains a boolean expression, *e.g.*, $R(x) ::= (x \geq 0)$.

Example 4. Applying the algorithm on the STGs in Figure 3(b) (implementation) and in Figure 2(a) (specification), we retrieve the following PES:

$$
\begin{array}{ll}
R_{1,1}() & ::= \forall Z_0\ R_{2,2}(Z_0, Z_0) \\
R_{2,2}(y_1, x_1) & ::= (((x_1 \geq 5 \Rightarrow y_1 \geq 5 \wedge R_{3,2}(y_1, x_1)) \wedge (y_1 \geq 5 \Rightarrow x_1 \geq 5 \wedge R_{3,2}(y_1, x_1))) \\
 & \quad \wedge ((x_1 < 5 \Rightarrow y_1 < 5 \wedge R_{4,3}) \wedge (y_1 < 5 \Rightarrow x_1 < 5 \wedge R_{4,3}))) \wedge (\neg(y < 10)) \\
R_{3,2}(y_1, x_1) & ::= ((x_1 \geq 5 \Rightarrow y \geq 10 \wedge R_{4,3}) \wedge (y \geq 10 \Rightarrow x_1 \geq 5 \wedge R_{4,3})) \\
 & \quad \wedge ((\neg(y < 10)) \wedge (\neg(x_1 < 5))) \\
R_{4,3}() & ::= \mathit{true}
\end{array}
$$

Indeed, it can be simplified into $\{R_{1,1} ::= y \geq 10, R_{2,2} ::= y \geq 10, R_{3,2} ::= y \geq 10 \wedge Z_0 \geq 5, R_{4,3} ::= \mathit{true}\}$ but this demonstrates the need for an automatic PES satisfiability checking procedure as defined below.

PES satisfiability and conformance verdict. The PES resulting from the conformance computation algorithm has to be analyzed in order to reach a conformance verdict. We realize this step with the Z3 SMT Solver by translating the PES into the Z3 input language as demonstrated in Listing 1.1 for the PES in Example 4. Each predicate equation in the PES is translated as a boolean function (using *define-fun*) and each free variable is translated as an integer function (using *declare-fun*). We then check `R1_1` in order to conclude on conformance. For this, the `check-sat` Z3 command is run as following. If `R1_1` asserted *false* (as in Listing 1.1) yields an *unsat* response then there is no interpretation such that $R_{1,1}$ is false, hence we can conclude directly that conformance is *true*. Otherwise, we have to retry with `R1_1` asserted to *true* to reach a verdict. The result may then be *unsat*, *sat*, or *timeout* corresponding respectively to the conformance being *false*, *may be* (ρ), or *inconclusive*.

Listing 1.1. Translation into the Z3 language of the PES in Example 4

```
1 (set-option :print-warning false)
2 (declare-fun y () Int)
3 (define-fun R4_3 () Bool true)
4 (define-fun R3_2 ((y_1 Int)(x_1 Int)) Bool (and (and (implies (>= x_1 5)
    (and (>= y 10) R4_3)) (implies (>= y 10) (and (>= x_1 5) R4_3))) (and (
    not (< y 10)) (not (< x_1 5)))))
5 (define-fun R2_2 ((y_1 Int)(x_1 Int)) Bool (and (and (and (implies (>=
    x_1 5) (and (>= y_1 5) (R3_2 y_1 x_1))) (implies (>= y_1 5) (and (>=
    x_1 5) (R3_2 y_1 x_1)))) (and (implies (< x_1 5) (and (< y_1 5) R4_3))
    (implies (< y_1 5) (and (< x_1 5) R4_3)))) (not (< y 10))))
6 (define-fun R1_1 () Bool (forall ((Z_0 Int)) (R2_2  Z_0 Z_0)))
7 (assert (= R1_1 false))  ; uncomment for step 1, comment for step 2
8 ; (assert (= R1_1 true)) ; comment for step 1, uncomment for step 2
9 (check-sat)
```

Experiments. We have experimented our framework, including on examples from the literature (Tab. 2). For the implementations and the specifications, we respectively give the numbers of peers, roles, interactions, and transitions and states in the corresponding STGs. We also give the conformance verdicts in the paper the example is taken from and with our approach. Finally, we give the execution time (Mac Book Air with OS 10.7, 4 GB RAM, core i5 1.7 GHz) for the process described in Figure 1 (but for the time to parse the input files). Rows 1 to 3 correspond to the specification STG in Figure 2(a) and, respectively, to the implementations STGs in Figures 2(d) (row 1), 3(a) (row 2), and 3(b) (row 3). Rows 4 and 5 correspond to the example and mutation in [6]. The difference in the verdict comes from the fact the we distinguish between an STG ending with ✓ (successful termination) or not, hence an implementation deadlocking after achieving all interactions of a specification will not conform to it: the specification may do ✓ while the implementation may not. Row 6 corresponds to a negative example in [7] and row 7 to a positive one in [4].

3 Conclusion

In this paper, we have proposed a formal framework for checking the conformance of a set of role requirements or peer descriptions with reference to a choreography specification. Symbolic models and equivalences enable us to check conformance in presence of data without suffering from state space explosion and without bounding data domains. Going further than strict conformance, we are able to

Table 2. Experimental results

Id	Name [Reference]	#Peers/Roles	Implementation		Specification		Verdict	Duration
			#Int.	#Trans./States	#Int.	#Trans./States	Orig./Ours	(seconds)
01	Shipping [n/a]	2/2	3	4/4	3	4/4	-/YES	0.069
		2/2	4	5/5	3	4/4	-/YES	0.084
	Example 4 →	2/2	4	6/5	3	4/4	-/ρ	0.102
04	Market [6]	4/4	8	9/10	8	10/10	YES/NO	0.118
		8/4	16	27/26	8	10/10	YES/NO	0.201
06	RFQ [7]	3/3	6	8/7	6	8/8	NO/NO	0.078
07	Booking [4]	4/4	8	12/11	8	12/11	YES/YES	0.096

give the most general constraint over data exchanged between peers in order to achieve conformance. Our approach is fully tool supported[3].

We advocate that once a choreography projection function supporting data is defined, then our framework could be used not only for conformance checking but also for realizability checking. This is our first perspective. A second perspective is to extend our framework with non-limited assignment and asynchronous communication. Our last perspective is to integrate the extensions of our tools as a verification plugin for the BPMN 2.0 Eclipse editor. A BPMN 2.0 to STG model transformation is ongoing, based on our BPMN to LTS (no data) one [17].

References

1. Poizat, P.: Formal Model-Based Approaches for the Development of Composite Systems. Habilitation thesis, Université Paris Sud (November 2011) http://www.lri.fr/~poizat/documents/hdr.pdf.
2. Qiu, Z., Zhao, X., Cai, C., Yang, H.: Towards the Theoretical Foundation of Choreography. In: Proc. of WWW '07. (2007)
3. Basu, S., Bultan, T.: Choreography Conformance via Synchronizability. In: Proc. of WWW'11. (2011)
4. Salaün, G., Bultan, T.: Realizability of Choreographies using Process Algebra Encodings. In: Proc. of IFM'09. (2009)
5. Li, J., Zhu, H., Pu, G.: Conformance Validation between Choreography and Orchestration. In: Proc. of TASE'07. (2007)
6. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration Conformance for System Design. In: Proc. of COORDINATION'06. (2006)
7. Kazhamiakin, R., Pistore, M.: Choreography Conformance Analysis : Asynchronous Communications and Information Alignment. In: Proc. of WS-FM'06. (2006)
8. Van Glabbeek, R., Weijland, W.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM **43**(3) (1996)
9. Hennessy, M., Lin, H.: Symbolic Bisimulations. Theoretical Computer Science **138**(2) (1995) 353–389
10. Lin, H.: Symbolic Transition Graph with Assignment. In: Proc. of CONCUR'96. (1996)
11. Li, Z., Chen, H.: Computing Strong/Weak Bisimulation Equivalences and Observation Congruence for Value-Passing Processes. In: Proc. of TACAS'99. (1999)
12. Basu, S., Mukund, M., Ramakrishnan, C., Ramakrishnan, I., Verma, R.: Local and Symbolic Bisimulation Using Tabled Constraint Logic Programming. In: Logic Programming. (2001)
13. Bravetti, M., Zavattaro, G.: Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In: Proc. of SC'07. (2007)
14. Nguyen, H.N., Poizat, P., Zaïdi, F.: A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies. Long version, in P. Poizat Webpage
15. Decker, G., Kopp, O., Barros, A.P.: An Introduction to Service Choreographies. Information Technology **50**(2) (2008) 122–127
16. Kopp, O., Leymann, F.: Do We Need Internal Behavior in Choreography Models? In: Proc. of ZEUS'09. (2009)
17. Poizat, P., Salaün, G.: Checking the Realizability of BPMN 2.0 Choreographies. In: Proc of SAC'12. (2012)