HAL
open science

VerChor: A Framework for Verifying Choreographies

Matthias Güdemann, Pascal Poizat, Gwen Salaün, Alexandre Dumont

To cite this version:

Matthias Güdemann, Pascal Poizat, Gwen Salaün, Alexandre Dumont. VerChor: A Framework for Verifying Choreographies. Fundamental Approaches to Software Engineering 2013, Mar 2013, Rome, Italy. pp.226-230, 10.1007/978-3-642-37057-1_16 . hal-00806788

HAL Id: hal-00806788

https://inria.hal.science/hal-00806788

Submitted on 2 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

VerChor: A Framework for Verifying Choreographies

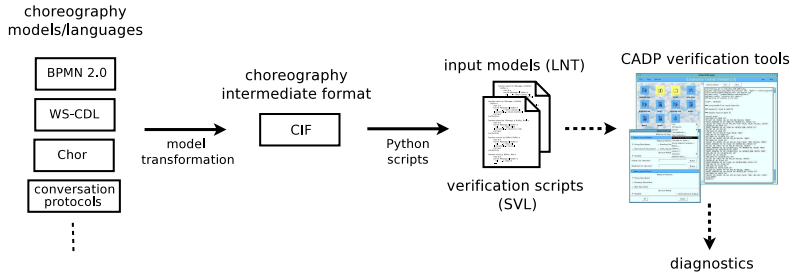
Matthias Güdemann[1], Pascal Poizat[2], Gwen Salaün[1], and Alexandre Dumont[1]

[1] Inria, Grenoble INP, France
[2] LIP6 UMR 7606 CNRS – Université Paris Ouest, Nanterre, France

1 Introduction

Nowadays, modern applications are often constructed by reusing and assembling distributed and collaborating entities, *e.g.*, software components, Web services, or Software as a Service in cloud computing environments. In order to facilitate the integration of independently developed components (*i.e.*, peers) that may reside in different organizations, it is necessary to provide a global contract to which the peers participating in a service composition should adhere. Such a contract is called *choreography*, and specifies interactions among a set of services from a global point of view. This contract is the reference for the further development steps (service selection, code generation, maintenance, reconfiguration, etc.). The specification and formal analysis of this contract is therefore crucial and must be handled carefully by the designer to avoid an erroneous design, which would be very costly if discovered lately in the development process. Unfortunately, only limited effort, *e.g.* [3, 6, 1], has been spent to develop formal verification tools, which can automatically detect issues such as deadlocks or erroneous behaviours in the choreography specification.

In this paper, we propose a modular framework for performing automatically a number of crucial verification tasks on choreography specifications. Our framework accepts as input the following interaction-based choreography description languages: (i) XML-based languages (WS-CDL), (ii) graphical notations (BPMN 2.0 choreographies), and (iii) formal description models (Chor, conversation protocols). In order to favour extensibility and reusability of our framework, we

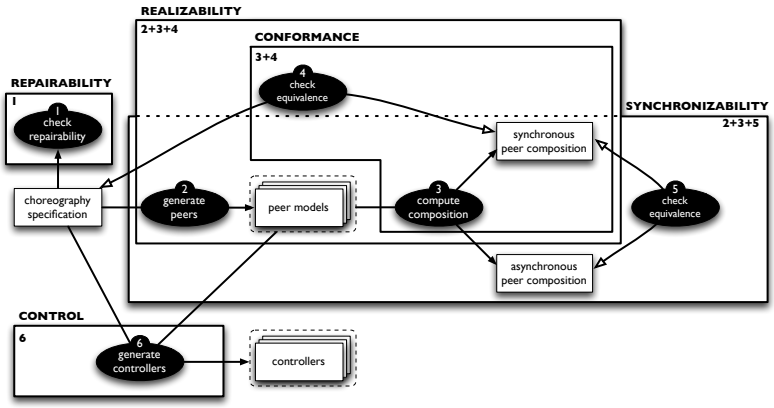


propose an intermediate format (CIF) for representing choreography description languages. This intermediate format allows to accept several existing languages

as input. It can also serve as an expressive standalone specification language for choreographies and can be easily extended with new choreography constructs. Another advantage is that it makes possible to use jointly several formal verification tools and techniques as back-end, provided that a connection to those tools exist. We have already developed a connection to the CADP verification toolbox [2] via a translation to the LNT process algebra, one of the CADP input specification languages. This enables the automated verification of some key choreography analysis tasks (repairability, realizability, conformance, etc.). Our framework is extensible with other front-end and back-end connections to, respectively, other choreography languages and formal verification tools.

2 Verification

This section presents some key properties, which are of utmost importance when designing choreography-based distributed system. They can be verified automatically in our framework using model and equivalence checking techniques.



Repairability. A choreography is not repairable when at some point in its behaviour there is a non-deterministic choice between interactions involving different sending peers. Such a design is erroneous because there is no way to make the corresponding distributed implementation respect the choreography requirements. Detecting automatically (non-)repairable choreographies is difficult, because there are situations where such a non-deterministic choice actually corresponds to the initial part of an interleaving of several interactions, and in that case, the choreography is repairable.

Synchronizability. Synchronizability analyzes a set of peers and checks that all interaction sequences in the asynchronous system are also possible in the synchronous one. This property is necessary for ensuring the realizability and conformance of possibly infinite systems (choreographies with loops). A recent result [1] that we reuse here proves that checking synchronizability is decidable and proposes a decision procedure for verifying this property.

Realizability. This property checks whether the distributed version of the system behaves exactly as specified in the choreography. This is crucial in a top-down development process, where peers are obtained via projection [6] from the choreography, in order to ensure that the implementation perfectly matches the global specification. In our framework, we can check equivalence-based notions of realizability, as those used in [1, 5, 6].

Conformance. In a bottom-up development process, peers are being reused and integrated into a new composition. The choreography serves as a contract that the implementation under construction must respect. From a verification point of view, it can be checked exactly as realizability, except that projection is not necessary. Conformance checking takes as input a choreography and a set of peers, whereas realizability checking only requires a choreography specification.

Control for enforcing realizability. If a choreography is not realizable (or conformant *wrt.* a set of peers) yet repairable, we can enforce the distributed system to respect the (synchronizability and) realizability of a choreography by generating distributed controllers. They act locally by interacting with their peer and the rest of the system in order to make the peers respect the choreography requirements. These controllers are generated through an iterative process, automatically refining their behaviours, as presented in [4].

3 Tool Support

We use Eclipse Indigo and the BPMN2 modeler as front-end for BPMN, and XML for describing our intermediate format. The connection from our intermediate format to CADP, that we use here for verifying the properties introduced in Section 2, is achieved through a translation to the LNT process algebra (see [5] for encoding patterns). CADP model and equivalence checking tools are used here for verifying automatically all the properties presented in Section 2. Verification of the properties is fully automated thanks to verification scripts generated by our translator. The encoding into LNT also enables other kinds of formal analysis with CADP, such as deadlock search, simulation, or checking temporal properties written in MCL using the Evaluator 4.0 model checker.

Experiments. We show experimental results on some examples of our database, which contains more than 200 choreographies (many of them are real-world examples found in the literature). It is worth observing that translation time (from the input languages to CIF and LNT) is negligible even for huge examples. For each experiment, the table gives the number of peers (P), interactions (Inter.), and selection operators (Sel.). Then, we give the size of the corresponding LTS and the size of the largest intermediate state space for generating the asynchronous version of the distributed system (number of states and transitions), and the overall time for checking whether the choreography is repairable (R_p), generating all LTSs (synchronous and asynchronous versions of the distributed system), and verifying synchronizability (S_c) and realizability (R).

We can see that BPMN choreographies can result in huge LTSs (see example 7), because BPMN parallel operators are expanded in all the possible inter-

Ex.	Lang.	|P|	|Inter.|	|Sel.|	$\|S\|/\|T\|$	Async. parallel compo. $\|S\|/\|T\|$	Time	Verif. R_p | S_c | R
1	Chor	3	10	1	21 / 29	127 / 200	48s	√ | √ | √
2	BPMN	6	19	1	580 / 1,828	4,054 / 12,814	1m43s	√ | √ | √
3	BPMN	6	19	1	18 / 20	750 / 3,298	1m40s	√ | √ | √
4	BPMN	6	19	1	580 / 1,842	16,129 / 51,317	1m45s	√ | √ | √
5	CP	7	11	1	11 / 11	158,741 / 853,559	5m47s	√ | × | ×
6	BPMN	12	25	4	577 / 2,499	$\sim 1*10^6$ / $\sim 7*10^6$	8m43s	√ | √ | √
7	BPMN	15	31	5	65,556 / 573,479	$\sim 2*10^6$ / $\sim 18*10^6$	1h34m	√ | × | ×

leaved behaviours when the corresponding LTS is generated. We note that the overall time for generating LTSs for choreography and both distributed systems (synchronous and asynchronous) as well as for verifying properties R_p, S_c, and R is reasonable for medium-size choreographies, see for instance examples 2, 3, 4, 6 in the table. It is most costly to check realizable examples because it deserves an exhaustive exploration of all cases, whereas if the choreography is not realizable, the analysis stops when a violation is found. The two causes of explosion are the number of peers (*e.g.*, 15 peers in example 7) and the degree of parallelism, that is the number of branches and interactions executed in concurrent branches of the choreography. If the choreography is not realizable, we generate local controllers which synchronize together in order to enforce the distributed system to respect the order of messages as specified in the global contract. For instance, example 5 presents several ordering issues if peers are generated using projection. In that case, our process requires 6 iterations to construct these controllers, meaning that 6 additional synchronization messages are necessary to make the system realizable. It takes about 20 minutes for this example to successively check synchronizability/realizability using equivalence checking and exploit the resulting counterexample to refine controllers, until completion of the process.

Acknowledgements. This work is supported by the Personal Information Management through Internet (PIMI) ANR project.

References

1. S. Basu, T. Bultan, and M. Ouederni. Deciding Choreography Realizability. In *Proc. of POPL'12*.
2. H. Garavel et al. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of TACAS'11*.
3. H. Foster, S. Uchitel, J. Magee, and J. Kramer. LTSA-WS: A Tool for Model-based Verification of Web Service Compositions and Choreography. In *Proc. of ICSE'06*.
4. M. Güdemann, G. Salaün, and M. Ouederni. Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In *Proc. of ATVA'12*.
5. P. Poizat and G. Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC'12*.
6. G. Salaün, T. Bultan, and N. Roohi. Realizability of Choreographies Using Process Algebra Encodings. *IEEE T. Services Computing*, 5(3), 2012.