Checking the Realizability of BPMN 2.0 Choreographies
Pascal Poizat, Gwen Salaün

To cite this version:
Pascal Poizat, Gwen Salaün. Checking the Realizability of BPMN 2.0 Choreographies. 27th Symposium On Applied Computing (SAC 2012), Mar 2012, Italy. pp.1927-1934, 2012. <hal-00685393>

HAL Id: hal-00685393
https://hal.archives-ouvertes.fr/hal-00685393
Submitted on 5 Apr 2012

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.
Checking the Realizability of BPMN 2.0 Choreographies

Pascal Poizat
University of Evry Val d’Essonne, Evry, France
LRI UMR 8623 CNRS, Orsay, France
pascal.poizat@lri.fr

Gwen Salaün
Grenoble INP, INRIA, France
gwen.salaun@inria.fr

ABSTRACT

Choreographies allow business and service architects to specify with a global perspective the requirements of applications built over distributed and interacting software entities. While being a standard for the abstract specification of business workflows and collaboration between services, the Business Process Modeling Notation (BPMN) has only been recently extended into BPMN 2.0 to support an interaction model of choreography, which, as opposed to interconnected interface models, is better suited to top-down development processes. An important issue with choreographies is realizability, i.e., whether peers obtained via projection from a choreography interact as prescribed in the choreography requirements. In this work, we propose a realizability checking approach for BPMN 2.0 choreographies. Our approach is formally grounded on a model transformation into the LOTOS NT process algebra and the use of equivalence checking. It is also completely tool-supported through interaction with the Eclipse BPMN 2.0 editor and the CADP process algebraic toolbox.

1. INTRODUCTION

Modern applications are no longer built as monolithic, stand-alone, programs. Rather, they are constructed out of the reuse and assembly of distributed and collaborating entities, e.g., business protocols, software components, or services. Software engineering processes have adapted to this, with two different viewpoints over application design and implementation. A first one is orchestration-oriented (Fig. 1). Given a set of services to be reused and the specification of the application-to-be in a centralized form, one retrieves a model of the orchestration and then implements it. Related issues are how to compose, and possibly adapt, reused services [21, 22] and then how to check implementation correctness [4]. This is a bottom-up approach and local perspective of service composition. However, this is not always desirable. Abstraction in the specification process promotes that the specification of a distributed application is performed with a global perspective on the interactions between the entities that compose it. This is the choreography-oriented viewpoint (Fig. 2).

Conformance vs. realizability. In a first case, the question is whether a set of peers being reused satisfies or not the choreography specification. This is conformance checking (Fig. 2, (A)) and is again a bottom-up approach of service composition. Let \( P_i \) denote the behaviour of some peer \( i \), \( || \) denote parallel composition of the peers (including communication), and \( C \) denote the expected behaviour. The conformance of a set of peers with a choreography corresponds to checking equation:

\[
P_1 || \ldots || P_i || \ldots || P_n \models C \tag{1}
\]

with all \( P_i \) and \( C \) being given. Different authors have proposed techniques related to this, using different semantics for communication (\( || \)) and equivalence (\( \models \)), e.g., [8, 19, 3],

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

\( \text{ACM'12 March 25-29, 2012, Riva del Garda, Italy.} \)

Copyright 200X ACM X-XXXXXX-XX-XX/XX ...$10.00.
depending on the properties to be preserved between the expected behaviour and the peer composition. However, these techniques suppose that the peers are available, which is not suitable in a top-down approach. In the second case (Fig 2, (1,2,3)), the question is whether the choreography specification can be implemented correctly (i.e., keeping the same behaviour) by projecting it on each peer. This is realizability checking [7]. In the case that it is not, there is no need to further advance in the design process with discovery, selection, and composition of services for peers. Information should be returned to the service architect in order to enable choreography refactoring. This corresponds to checking Equation (1) again but with each \( P_i \) being the projection of \( C \) on peer \( i \).

Let us take the choreography in Figure 3, left, described with a Choreography Diagram as introduced by the BPMN 2.0 standard (see Sect. 2 for an introduction and motivation to BPMN 2.0). It specifies that \( P_1 \) and \( P_2 \) must first interact on message \( M_1 \) (with \( P_1 \) being the initiator of the interaction) and then \( P_3 \) and \( P_4 \) will interact on message \( M_2 \) (with \( P_3 \) being the initiator). It is not realizable since there is no possibility for \( P_3 \) to know that \( P_1 \) has sent (or \( P_2 \) has received) message \( M_1 \) before sending \( M_2 \). In a distributed setting, implementing \( P_1 \) (resp. \( P_3 \)) as sending \( M_1 \) (resp. \( M_2 \)) we could have the case where \( M_2 \) is sent before \( M_1 \), which is prevented by the choreography. Another issue is that realizability depends on the kind of communication. Let us take the choreography in Figure 3, right, and suppose that \( P_3 \) sends \( M_2 \) and that \( P_1 \) is implemented as first sending \( M_1 \) and then receiving \( M_2 \). In a synchronous setting, the choreography is realizable since \( P_3 \) will be blocked upon sending \( M_2 \) until \( P_1 \) is ready to receive it. However, in an asynchronous setting, we would have the same problem as above.

![Figure 3: Examples of unrealizable choreographies](image)

**Contributions.** The issue addressed in this paper is the following: given the specification of a choreography described in the BPMN 2.0 standard notation, does there exist a possible implementation of it, i.e., peer implementations obtained by projection from it, that would perform exactly as prescribed in the specification. Our first contribution is the transformation of BPMN 2.0 choreographies into a process algebra, namely LOTOS NT [10]. This paves the way for the development of formal tools dedicated to the BPMN 2.0 choreography notation, such as model-checking, testing, automatic composition, and software adaptation. Our second contribution is related to realizability. We propose techniques to check the realizability of BPMN 2.0 choreographies, thus releasing the burden on service architects when structural well-formedness constraints are required to achieve realizability. With our approach, the architect is totally free to design any desired choreography. Realizability analysis is automatically achieved as a subsequent step. Furthermore, we study realizability in both synchronous and asynchronous communication frameworks, since the kind of communication has an important impact on realizability.

This fosters the applicability of the techniques we propose. Finally, our approach is totally tool-supported, bridging design (achieved using the Eclipse BPMN 2.0 editor) and verification (achieved using the CADP verification toolbox [16]).

**Organization.** The rest of the paper is organized as follows. In Section 2 we advocate the use of BPMN 2.0 for the specification of choreographies and present the related notations. We also make a short introduction to the LOTOS NT process algebra. The next three sections detail our approach for BPMN 2.0 choreography analysis. Section 3 presents our formal model transformation from BPMN 2.0 choreographies to LOTOS NT processes. Section 4 explains how this encoding can be used to analyze choreographies and check their realizability, and tool support is addressed in Section 5. In Section 6 we discuss related work and compare our approach to it. Finally, Section 7 summarizes our contributions and presents perspectives of our work.

## 2. BPMN 2.0 Choreographies and LOTOS NT

### 2.1 BPMN 2.0 Choreographies

The Business Process Modeling Notation, BPMN [25], is a visual notation that well suits the needs for the specification of business processes and their interactions, including future service implementations. With reference to low-level notations or languages such as the Web Service Business Process Execution Language (WS-BPEL), BPMN promotes a high-level description that enables the service architects to focus on what services do in a composition, possibly including their conversations and interactions, without entering too much into the details of how they do it.

As classified by [12], there are two interaction models for choreography: interconnected interface models, where conversations are defined at (each) peer level and interactions are defined by roughly connecting conversations, and interaction models, where interactions between peers are the basic building blocks. The former nicely suits low-level languages such as WS-BPEL where an orchestration would be defined for each peer and communications would correspond to connections between peer models. However, from a designer perspective, and following the separation of concerns principle, interaction models better suit the needs of choreography specification due to their global perspective.

A drawback of BPMN 1.x was that it supported choreography specifications through an interconnected interface model only (Collaboration Diagrams). In contrast, the Web Service Choreography Description Language (WS-CDL) enables a real interaction model for choreography descriptions, i.e., the basic building blocks of choreographies are peer interactions. However, WS-CDL is more an implementation language than a specification notation. The lack of a standard abstract notation supporting an interaction model for choreography has been remedied with the introduction of BPMN 2.0. In addition to Collaboration Diagrams, BPMN 2.0 introduces Choreography Diagrams to support conversations with choreography tasks as a first class entity.

The basic building block of BPMN 2.0 (BPMN in the rest of this paper) Choreography Diagrams is a one-way or two-way interaction between peers. This is modelled using a choreography task (Fig. 4). Here we have two peers interacting, A and B, represented by participant bands. A is
the initiating peer, i.e., the one that decides the interaction takes place, hence it is represented by a white band as opposed to a gray filled band for B. Together with the choreography tasks, come message flows relating the interaction with an initiating message (represented by a white envelope) and, possibly, a return message (represented by a black envelope). This yields one-way interactions (Fig. 4, (a,c,d,e)) or two-way interactions (Fig. 4, (b)).

A choreography task may have an internal marker to denote whether, and how, the related interaction (one or two message exchanges) is repeated. In a standard loop (Fig. 4, (c)), the interaction is performed several times depending on a Boolean condition. In multi-instance parallel loops, the interactions are performed by several instances of the choreography task. This can be done in parallel (Fig. 4, (d)) or in sequence (Fig. 4, (e)). If the exchange is not repeated, no marker is used (Fig. 4, (a,b)).

BPMN enables one to describe control flows using sequence flows for performing two tasks in sequence or gateways for more complex behaviours. In our work we take into account the main gateways found in BPMN (Fig. 5) that is: exclusive gateways (decision, alternative paths), inclusive gateways (inclusive decision, alternative but also parallel paths), parallel gateways (creation and merging of parallel flows), and event-based gateways (choice based on events, i.e., message reception or timeout, like the WS-BPEL pick construct). We require that gateways are either diverging (multiple outgoing sequence flows and at most one incoming sequence flow) or converging (multiple incoming sequence flows and at most one outgoing sequence flow). Diagrams that would not cope to this requirement can be transformed by adding new gateways [25], e.g., a gateway being both converging and diverging can be transformed as the sequence of a converging one and a diverging one.

![Figure 5: BPMN 2.0 notation – control flow and gateways](image)

**Running example.** In this paper, we will use an e-booking system as running example. This case study involves four peers: a booking system (bs), a database (db), an on-line bank service (bk), and a client (cl). Figure 6 gives a BPMN choreography for the system under construction. In this specification, we can see that first the client interacts with the booking system (connect), submits a request to the booking system (request) and receives a response (reply). Then, the client can either choose to quit (abort), submit another request (request), or make a booking (book). In this last case, the client also pays the bank (pay), and the booking system stores some information in the database to keep track of this completed transaction (store).

![Figure 6: Running example – Booking System](image)

### 2.2 LOTOS NT

LOTOS NT [10] is an improved version of the LOTOS ISO standard [17] that combines the best features of imperative programming languages and value-passing process algebras. LOTOS NT also supports the description of complex data types written using a functional specification language. LOTOS NT has a user-friendly syntax and formal operational semantics defined in terms of Labeled Transition Systems (LTSs).

LOTOS NT processes are built from actions, sequential compositions (;), conditions (if), assignments (:=), looping behaviours (loop), choices (select), and parallel compositions (par). Communication is carried out by rendezvous on a set of synchronization actions (multiway synchronization points) with bidirectional transmission of multiple values. Synchronizations may also contain optional guards (where) expressing Boolean conditions on received values. Processes are parameterized by sets of actions (alphabets) and input/output data variables. In the rest of this paper, we will show examples of LOTOS NT generated from BPMN choreographies for realizability checking purposes.

LOTOS NT specifications can be analysed using the CADP verification toolbox [16]. In particular, LOTOS NT is supported by the int.open tool of CADP, which enables the on-the-fly exploration and verification of the LTSs corresponding to LOTOS NT specifications.

### 3. ENCODING INTO LOTOS NT

In this section, we present the encoding of BPMN choreographies into LOTOS NT. We chose LOTOS NT, instead of, e.g., Petri nets encodings or a direct translation into LTSs, for several reasons. First, LOTOS NT provides expressive operators for translating BPMN constructs, and a translation between two high-level languages is much simpler because both languages share similar operators, such as sequence, choice, interleaving, etc. Second, LOTOS NT is supported by state-of-the-art verification tools (CADP) that
can be used in order to compile the LOTOS NT specification into an LTS, enumerating all the possible behaviours. The LOTOS NT operators together with the CADP tools enable us, for instance, to generate peers (Fig. 2, (2)) in a declarative way, thus avoiding the implementation of low-level ad-hoc algorithms operating on LTS models.

We encode BPMN choreographies in LOTOS NT following the state machine pattern. Each BPMN construct is encoded as a LOTOS NT process that symbolizes a state in that state machine. This process translates the behaviour of the BPMN construct plus a call to the process encoding the successor state (possibly several process calls if there are several successors, e.g., in the case of a diverging gateway). The initial choreography element identified by a start state in BPMN (e.g., CT1 in Fig. 6) is translated as a LOTOS NT process called MAIN.

We show below the LOTOS NT process encoding the second activity of our running example. This process is named CT2 and makes explicit the alphabet between squared brackets, that is, all the different messages exchanged in the choreography. These messages can be typed (with the types of sets, that is, all the different messages exchanged in the choreography). These messages can be typed (with the types of sets, that is, all the different messages exchanged in the choreography. The initial choreography element identified by a start state in BPMN (e.g., CT1 in Fig. 6) is translated as a LOTOS NT process called MAIN.

We show below the LOTOS NT process encoding the second activity of our running example. This process is named CT2 and makes explicit the alphabet between squared brackets, that is, all the different messages exchanged in the choreography. These messages can be typed (with the types of sets, that is, all the different messages exchanged in the choreography. The initial choreography element identified by a start state in BPMN (e.g., CT1 in Fig. 6) is translated as a LOTOS NT process called MAIN.

The generated LOTOS NT code is as follows:

```plaintext
process INCL_1 [A_B_M1: any, A_C_M2: any]
select
CT1 [A_B_M1: any, A_C_M2: any] ; CT2 [..]
[] CT2 [A_B_M1: any, A_C_M2: any] ; CT1 [..]

CT1 [A_B_M1: any, A_C_M2: any]
[] CT2 [A_B_M1: any, A_C_M2: any]
end select
end process
```

We recall that diverging inclusive gateways may fire one or more branches. Consequently, this operator is translated as a choice (select) between all possible combinations of the subsequent activities involved in the gateway. Suppose for instance that we have an inclusive gateway between two communications M1 and M2 involving three peers A, B, and C (Fig. 7, left).

![Figure 7: Examples of diverging inclusive gateway (left), diverging and converging parallel gateways (right)](image)

The encoding of the choreography as a state machine simplifies the translation of looping behaviours, e.g., in our example submitting as many requests as the client decides. The drawback is that some structuring information gets lost, in particular in order to translate converging gateways; one no longer knows which branches are executed in parallel and need to be merged. Therefore, to translate correctly converging/diverging gateways, we compute and generate further information. This only concerns parallel and inclusive gateways where several branches can be executed at the same time, and for which we may need to generate a proper merge (a single branch is executed in an exclusive gateway).

Let us illustrate this with a simple example where we have successively a diverging parallel gateway followed by two interactions M1 and M2, and a converging parallel gateway (Fig. 7, right). In such a case, we need a synchronization point to merge the different branches involved in this gateway. To do so, when we translate the diverging gateway, we traverse the specification forward to detect a possible converging gateway. If there is a converging gateway, we generate an additional branch in the LOTOS NT parallel composition with a synchronization on an arbitrary action sync_1. After this action, we translate the rest of the choreography (process PAR_2 below). Once processes CT1 and CT2 have completed their behaviour, they can synchronize on sync_1 with the additional branch, and then process PAR_2 is executed. This synchronization is meaningless from an external point of view, it is why we hide it in process PAR_1.
behaviours. Third, we build the system composed of these encoding as presented above. Second, we generate peers' choreography L TS from the LOTOS NT cause L TS models for peers are minimized using the same it is unrealizable. Here, we use strong equivalence [24] be-

interactions as those specified in the choreography , therefore they do not generate the same peer generation of interacting peers; some additional FIFO buffers are necessary if an asynchronous communication model is assumed. Finally, we check that the choreography LTS and the distributed version of the system are equivalent.

Synchronous communication. In order to generate peers declaratively, we extend our LOTOS NT encoding by generating a LOTOS NT process for each peer. We show below the LOTOS NT process generated for the booking system. We can see that each peer process calls the MAIN process encoding the choreography in LOTOS NT as presented in Section 3. Moreover, we hide all messages in which the peer at hand is not involved, e.g., we hide the payment (cl_bk_pay) between the client and the on-line bank in the peer corresponding to the booking system (peer_bk):

Asynchronous communication. Asynchronous communica-
tion is not directly supported by LOTOS NT. Therefore, we generate additional LOTOS NT code to implement bounded FIFO buffers and support asynchronous communication while preserving the message ordering (since realizability is sensitive to it). Buffers are described using the functional part of the LOTOS NT specification language, and handled through specific processes. Each peer process is associated with a buffer process from which it can consume messages received beforehand. A buffer process can also receive messages from other peers, and store them. Note that a local communication between a peer and its buffer has the suffix “_REC”, whereas a communication between a peer (sender) and a buffer does not have a suffix. Finally, as for synchronous communication, we generate a process corresponding to the composition, which makes explicit the way peers and buffers interact.

This is demonstrated in Figure 9 for the database (db) peer. We also show below an excerpt of our running example showing how couples (peer,buffer) are generated for each peer. We see that the peer (peer_db) and its buffer (buffer_db) are composed in parallel (par) and can interact on bs_db_store_REC meaning that the peer can read this message from its buffer. The buffer has also the message

4. REALIZABILITY

From the LOTOS NT encoding of a BPMN choreography, we can generate the corresponding LTS model (Fig. 8) using the CADP state space exploration tools. This LTS was obtained by hiding “sync...” messages (which result in τ transitions), and by minimizing the resulting LTS using a weak trace reduction to remove τ transitions and determine the LTS, and a strong reduction to suppress identical paths. In

addition to these reduction techniques, all the CADP verification tools can be used to analyze this LTS. For instance, we can write and check temporal logic properties using the Evaluator model checker [23]. Concerning our running example, we can check that a client can make a booking or abort only if a request has been issued (safety property):

As for realizability, intuitively, a choreography is realizab-
le if the set of interactions specified in BPMN and those executed by the interacting peers (obtained by projection from the choreography description) are the same. In this paper, we propose to check the realizability by comparing the choreography model with the model of the system composed of interacting peers using behavioural equivalences. If these two models are equivalent, it means that the peer generation exactly preserves the BPMN communication requirements. If they are not, it is because peers do not generate the same interactions as those specified in the choreography; therefore it is unrealizable. Here, we use strong equivalence [24] because LTS models for peers are minimized using the same reductions as for the choreography LTS.

Realizability checking is performed in several steps. First, we generate the choreography LTS from the LOTOS NT encoding as presented above. Second, we generate peers' behaviours. Third, we build the system composed of these

Figure 8: Booking System – LTS model
Figure 9: System architecture: communication between peers and buffers

bs_db_store in its alphabet corresponding to the interaction between the booking system and the database buffer on that message:

```plaintext
process peer_buffer_db
    [bs_db_store: any, bs_db_store_REC: any] is
    par bs_db_store_REC in
    apeer_db [bs_db_store_REC]
    ||
    buffer_db [bs_db_store, ..] (bbuffer(nil,1))
end par
end process
```

Many verification problems are undecidable when asynchronous communication is assumed [5]. In our framework, the user can either choose bounds for buffers (1 in the example above), or check for synchronizability [3]: a set of services is synchronized if and only if the ordering of message exchanges remain the same when asynchronous communication is replaced with synchronous communication. Hence, if a system is synchronizable, the verification with CADP can be done on the synchronous version of the system and the results hold for the asynchronous case (without requiring any arbitrary bound).

Running example. Once we have generated all the required LOTOS NT processes, we use equivalence checking to compare on one hand the choreography LTS, and on the other the LTS generated for the distributed implementation of the system. As far as our running example is concerned, the equivalence check returns false for both communication models (synchronous and asynchronous), and indicates that the trace consisting of messages cl_bk_pay and bs_db_store are executed independently from one another in the distributed system, whereas the choreography specification imposes an order between these messages. Since these two interactions are independent, the simplest solution is to make it explicit in the BPMN choreography by using a diverging parallel gateway as shown in Figure 10. In such a case, our realizability check returns positive results whatever communication model is chosen.

5. TOOL SUPPORT AND EXPERIMENTS

Figure 11: Tool support

Our approach is fully tool-supported as shown in Figure 11. In order to foster the usability of the approach, we have built it on top of the BPMN 2.0 Editor Eclipse plug-in. Business process designers and service architects can use this plug-in to specify a BPMN choreography. A transformation from the XML representation of the choreography into an intermediary python format that we have developed is then performed using BPMN2Py, a .jar java runnable archive. BPMN2Py also calls Py2LNT, a python script that generates both the encoding of the choreography in LOTOS NT and a CADP SVL script that automates the remaining steps to be done for verification and realizability checking. The first step, i.e., the retrieval of LTS formal models for the peers, for their composition, and for the intended choreography behaviour, is achieved using the Caesar compiler and Reductor. The second step, i.e., verification, equivalence checking, and hence realizability, is achieved using Evaluator and Bisimulator. In case where realizability (or verification) does not yield, a counterexample is passed back to the designer in the form of a sequence of events x.y.m (x sends message m to y), enabling him/her to change the choreography. We have developed the BPMN2Py and Py2LNT tools, and we reuse the SVL script engine, and CADP state space generation and verification tools (model checker, equivalence checker, etc.).

Experiments. Table 1 shows experimental results on some examples of our database. Experiments have been carried out on a Xeon W3550 (3.07GHz, 12GB RAM) running Linux. For each experiment, the table gives the size of the BPMN choreography (in terms of number of peers, interactions, and gateways), the size of the LTS generated for the synchronous and asynchronous versions of the distributed system, the result of the synchronizability check, the realizability results for both communication models, and the overall execution time necessary to generate these LTSs and
compute these three checks. If the synchronizability prop-
erty is not satisfied, we use 1-bounded buffers to check the
choreography realizability. Example c0053 corresponds to
the running example presented in this paper. It is worth
observing that the number of peers involved in the com-
position as well as the number of parallel gateways used in the
choreography specification increases the parallelism degree
of the system and makes LTS sizes and computation times
augment quickly (c0102).

6. RELATED WORK

Quite some work has already been dedicated to the realiz-
ability issue. The results presented in [9, 26] formalise well-
formedness rules to enforce the specification to be realiz-
able. More precisely, in [9], Carbone et al. identify three prin-
ciples for global description under which they define a sound
and complete end-point projection that is the generation
of distributed processes from the choreography description.
In [26], Qiu et al. propose a choreography language with
new constructs (named dominated choice and loop) in order
to implement unrealizable choreographies. During the pro-
jection of these new operators, communications are added
in order to make peers respect the choreography specifica-
tion. However, these solutions only focus on synchronous
communication, and, similarly to what has been chosen for
BPMN 2.0 choreographies [25], complicate the design by
obliging the designer to make explicit extra-constraints in
the choreography specification, e.g., by associating domi-
nant roles to certain peers.

In [13], Decker and Weske propose a Petri Net-based for-
malism for specifying choreographies, and define realizabil-
ity and local enforceability. They also propose algorithms to
check these two properties. However, they consider only syn-
chronous communication, and have not investigated map-
plings to higher-level interaction modeling languages (such
as BPMN) yet.

Few works focused on the realizability problem assuming
asynchronous communication. Fu et al. [15] proposed three
sufficient conditions (lossless join, synchronous compatible,
autonomous) that guarantee a realizable conversation pro-
tocol. More recently, Basu and Bultan proposed to check
conformance using synchronizability [3]: A set of peers is
synchronizable if systems produced on one hand with syn-
chronous communication, and on the other with 1-bounded
asynchronous communication, are equivalent. If a set of
peers is synchronizable, one can check whether it is con-
formant to a choreography using existing finite state verifi-
cation tools. Synchronizability complements the techniques
proposed here, as presented in Section 4. In [6], Bravetti and
Zavattaro tackle the choreography conformance issue from
a theoretical point of view, and propose notions of contract
refinement and choreography conformance for services that
communicate through message queues.

Bultan and Fu [7] defined sufficient conditions to test real-
izability of choreographies specified with UML collabora-
tion diagrams (CDs). In [27], Salonin and Bultan refine and ex-
tend this work with techniques to enforce realizability by
adding additional synchronization messages among peers,
and a tool-supported approach to automatically check the
realizability of CDs for bounded asynchronous communica-
tion. The realizability problem for Message Sequence Charts
(MSCs) has also been studied (e.g., [1, 28, 2]). [2] presents
some decidability results on bounded MSC graphs, which are
graphs obtained from MSCs using bounded buffers. These
solutions are limited in the BPMN 2.0 context, because
branching and cyclic behaviours are not well supported by
CDs and MSCs (no choice in CDs, no cyclic behaviours in
MSCs, and only self-loops in CDs). BPMN choreographies
provide a more expressive notation than CDs or MSCs.

Decker and Weske present in [14] an extension of BPMN
(iBPMN) in the direction of interaction modeling. They also
propose a formal semantics for iBPMN in terms of interac-
tion Petri nets. At the end of this paper, the authors men-
tion realizability as a novel challenge, but do not give any
solution for this issue. Lohmann and Wolf [20] show how re-
alizability can be verified by using existing techniques for the
controllability problem, which checks whether a service has
compatible partner processes. They mention several mod-
els that can be used for modeling choreographies, such as
iBPMN, but present their results on multi-peer automata
called choreography automata. Their approach works for
peers interacting via arbitrary bounded buffers, and only
consider finite conversations. Here, we focused on the trans-
lation of high-level notations (BPMN) to a process algebra
(LOTOS NT) for formal analysis purposes. This encoding,
as well as the realizability check, are tricky issues, and even
error-prone if manually done. This is the reason why some
automated techniques and tools are absolutely essential.

7. CONCLUDING REMARKS

In this paper, we have focused on the part of the BPMN
notation dedicated to the choreography design. We have
proposed an encoding of a significative subset of BPMN
choreographies into the LOTOS NT process algebra. This
allows designers to validate their models using the CADP
verification tools. They can also check whether their chore-
ography model is realizable or not. If realizability is not
ensured, a counter-example is returned which identifies a
source of unrealizability. Our approach is fully automated
and has been validated on many examples.

As far as perspectives are concerned, we would like to
extend the subset of BPMN choreographies accepted by
our approach with hierarchical structuring aspects (sub-
choreography). In this work, we chose a strong notion of
realizability based on behavioural equivalence. Looser re-
alizability notions have been investigated [18], and we plan
to integrate them in our framework. We would also like
to use recent compositional aggregation techniques [11] to
reduce the intermediate state spaces size and improve the
realizability checking computation times. We would like to
propose a smart projection in order to enforce realizability.
If a choreography is unrealizable, we would automatically
generate peers extended with the minimum number of syn-
chronization points to make them respect the choreography
requirements. Finally, this work is a proof of concept on the
automatic checking of realizability for the new BPMN 2.0
choreography notation. Within the context of a project on
assisted electronic procedure fulfillment based on local (or-
chestration) and global (choreographic) contracts, we plan
to experiment with our approach on a real-size case-study
in the e-governance domain.

Acknowledgements. This work is supported by the Per-
sonal Information Management through Internet project
(PIMI-ANR-2010-VERS-0014-03) of the French National
Agency for Research.
Table 1: Realizability results for some case studies

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c0007</td>
<td>2</td>
<td>6/151</td>
<td>192/381</td>
<td>√</td>
<td>19s</td>
</tr>
<tr>
<td>c0025</td>
<td>3</td>
<td>5/112</td>
<td>143/358</td>
<td>√</td>
<td>24s</td>
</tr>
<tr>
<td>c0053</td>
<td>4</td>
<td>5/108</td>
<td>1.058/3,430</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>c0085</td>
<td>5</td>
<td>4/10</td>
<td>6.097/27,818</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>c0102</td>
<td>6</td>
<td>4/9</td>
<td>29,056/178,560</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

8. REFERENCES


