Synthesis of resilient choreographies

Marco Autili, Amleto Di Salle, and Massimo Tivoli *

Università degli Studi di L'Aquila, Italy
{marco.autili,amleto.disalle,massimo.tivoli}@univaq.it

Abstract. A possible Service Engineering (SE) approach to build service-based systems is to compose together distributed services by considering a global specification of their interactions, namely a choreography. BPMN2 (Business Process Modeling Notation v2.0) provides a dedicated notation, called Choreography Diagrams, to define the global expected behavior between interacting participants. An interesting problem worth considering concerns choreography realizability enforcement, while ensuring a resilient evolution upon facing changes. The strategy that we adopt to solve this problem is twofold: given a BPMN2 choreography specification and a set of existing services discovered as possible participants, (i) *adapt* their interaction protocol to the choreography roles and (ii) *coordinate* their (adapted) interaction so to fulfill the global collaboration prescribed by the choreography. This paper proposes a synthesis approach able to automatically generate, out of a BPMN2 choreography specification, the needed adaptation and coordination logic, and distribute it between the participants so to enforce the choreography. Our approach supports choreography evolution through adaptation to possible changes in the discovered services, while still keeping the prescribed coordination.

Keywords: Service Choreography, Model Driven Engineering, Service Oriented Architectures, Choreography Realizability Enforcement, Resilient Choreography Evolution

1 Introduction

Service-Oriented Computing (SOC) is now largely accepted as a well-founded reference paradigm for the Future Internet computing [16]. The near future in service-oriented system development envisions an ultra large number of diverse service providers and consumers that collaborate to fit users' needs. In this vision, a possible Service Engineering (SE) approach to build service-based systems is to compose distributed services together by considering a global specification of the interactions between the participant services, namely *Choreography*. Service choreographies will certainly have an important role in shaping the SOC within the vision of Future Internet. Choreography formalizes the way business participants coordinate their interactions. The focus is not on orchestrations of the work performed within them, but rather

* This work is supported by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet - www.choreos.eu).

on the exchange of messages between these participants. In this respect, a choreography defines the global expected behavior between interacting participants.

When considering choreography-based service-oriented systems, the following two problems are usually considered: (i) *realizability check* - checks whether the choreography can be realized by implementing each participant so that it conforms to the played role; and (ii) *conformance check* - checks whether the set of services satisfies the choreography specification. In the literature many approaches have been proposed to address these problems (e.g., [7,20,17,4,5]). However, by moving a step forward with respect to the state of the art, a further problem worth considering when actually realizing service choreographies by reusing (third-party) services concerns *automatic realizability enforcement*. That is, given a choreography specification and a set of existing services (discovered as suitable participants), externally coordinate their interaction so to fulfill the collaboration prescribed by the choreography specification. To address this problem, in this paper, we propose to (i) possibly adapt those services that have been discovered so to fit the choreography roles and (ii) synthesize the global coordination logic to be then distributed and enforced among the considered services. Discovery issues are out of scope of this paper. However it is worth mentioning that, whatever discovery process one wishes to apply, it is very infrequent to find a service that exactly matches the discovery query. For this reason, in the literature, many approaches have been devised (e.g., [1,26]) in order to account for an effective notion of similarity, which is an approximative notion. However, in the context of the CHOReOS EU project[1] where the full automation of the realizability enforcement process and its "resiliency" is of paramount importance, we cannot rely on services that approximately/partially play a choreography role. Thus, our approach is to synthesize and use adaptors in order to solve the problem of choreography realizability by enforcing exact similarity between the discovered services and the choreography roles. Adaptation to possible changes in the considered services is also a mean to achieve the realizability of resilient choreographies, i.e., choreographies able to evolve while still keeping the prescribed coordination.

We use BPMN2 to specify choreographies. The OMG BPMN2 [15] is the standard de facto for specifying service choreographies by providing a dedicated notation called *Choreography Diagrams*.

Contribution. In this paper we describe how to automatically synthesize a resilient choreography out of an its specification and a set of existing services. To this purpose, it is worth to note that, since a choreography is a network of collaborating services, the notions of *protocol adaptation* and *coordination protocol* become crucial. In fact, it might be the case that on the one hand the considered services do not exactly fit the choreography roles or changes in a participant service can be applied and, on the other hand, an uncontrolled collaboration of (possibly adapted) services can lead to *undesired interactions*. That is interactions that do not belong to the set of interactions modeled by the choreography specification. To prevent undesired interactions, we automatically synthesize additional software entities, called *Coordination Delegates* (CDs), and interpose them among the participant services. CDs coordinate the services' interaction in a way that the resulting collaboration realizes the specified choreography. This is done by exchanging suitable *coordination information* that is automatically generated out of

[1] See at `www.choreos.eu`.

the choreography specification. Furthermore to both adapt the interaction of the participant services so to fit the choreography roles and support choreography evolution, we automatically synthesized adaptors able to mediate the interaction service-CD and CD-service according to the specification of the corresponding choreography roles.

Progress beyond state-of-the-art. We tackle the problem of realizability enforcement, which so far has been receiving little attention by the SE community. Furthermore, our synthesis method allows the realizability of resilient choreographies that, to the best of our knowledge, are not accounted for by the state-of-the-art work.

Structure of the work. The paper is structured as follows. Section 2 and Section 3 describes the choreography synthesis process by means of basic examples and gives an intuition of how adaptation can be performed and correct coordination can be enforced. In Section 4 an explanatory example is then used to show the synthesis approach at work. Related works are discussed in Section 5. Section 6 concludes the paper and discusses future directions.

2 Choreography coordination synthesis

This section describes the synthesis approach and explains the notion of undesired interaction. The synthesis process uses dedicated model transformations to generate from a BPMN2 choreography diagram an automata-based specification of the coordination logic "implied" by the choreography. Specifically, an extension of Labelled Transition Systems (LTSs), called *Choreography LTS* (CLTS), is generated to explicitly describe the coordination logic that must be applied to enforce the choreography. CLTSs represent the mean to precisely describe the complex coordination logics implied by BPMN2 choreography specifications.

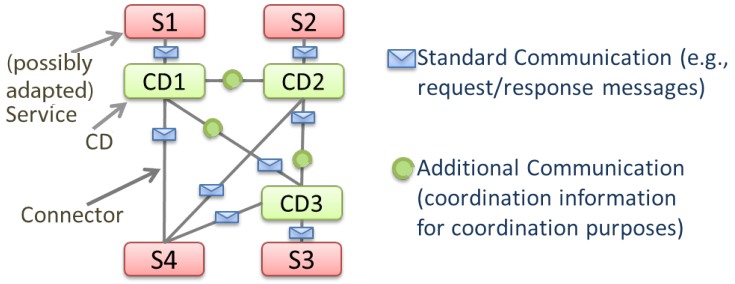

Fig. 1. CHOReOS Architectural Style

For the choreography to be externally enforced, the coordination logic modeled by the CLTS is distributed between additional software entities, whose goal is to coordinate (from outside) the interaction of the participant services in a way that the resulting collaboration realizes the specified choreography. To this aim, our method automatically derives these software entities, called *Coordination Delegates* (*CD*s), and interpose them among the participant services according to the CHOReOS architectural

style (see Fig. 1). CDs perform pure coordination of the services' interaction (i.e., *standard communication* in the figure) in a way that the resulting collaboration realizes the specified choreography. To this purpose, the coordination logic is distributed among a set of *Coordination Models* that codify coordination information. Then, at run time, the CDs exchange this coordination information (i.e., *additional communication*) to prevent possible undesired interactions. The latter are those interactions that do not belong to the set of interactions allowed by the choreography specification and can happen when the services collaborate in an uncontrolled way. In order to understand the notion of undesired interactions let us consider the very simple example in Fig. 2.

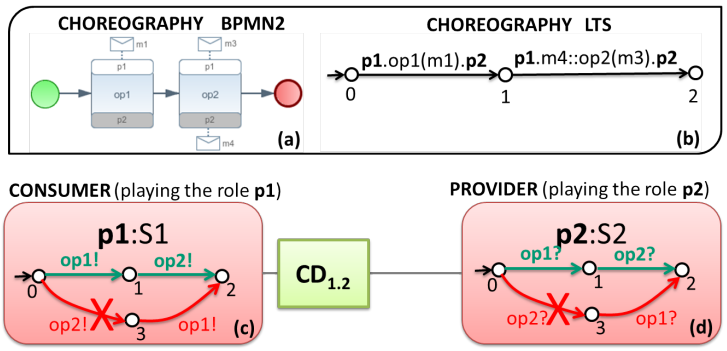

Fig. 2. Undesired interactions

The latter shows a BPMN2 choreography specification (a) and its corresponding CLTS (b). In BPMN2, a choreography `Task` is an atomic activity that represents an interaction by means of one or two (request and optionally response) message exchanges between two participants. Graphically, BPMN2 choreography diagrams uses rounded-corner boxes to denote choreography tasks. Each of them is labeled with the roles of the two participants involved in the task (see $p1$ and $p2$), and the name of the service operation (see $op1$ and $op2$) performed by the initiating participant and provided by the other one. A role contained in the white box denotes the initiating participant ($p1$ in the two tasks). In particular, we recall that the BPMN2 specification employs the theoretical concept of a token that, traversing the sequence flows and passing through the elements in a process, aids to define its behavior. The start event generates the token that must eventually be consumed at an end event. Basically, the BPMN2 model in the figure specifies that the task $op1$ is followed by the task $op2$. The corresponding CLTS models this sequence of tasks by specifying two contiguous transitions. In particular, the transition label $p1.m4{::}op2(m3).p2$ specifies that the participant $p1$ initiates the task $op2$ by sending the message $m3$ to the receiving participant $p2$ which, in turn, returns the message $m4$. Let us now assume that $S1$ and $S2$ are the services that have been discovered to play the roles of $p1$ and $p2$, respectively.

The automaton of $S1$ ($S2$) specifies that $S1$ ($S2$) initiates (receives) $op1!$ ($op1?$) as first, and initiates (receives) $op2!$ ($op2?$) as second, and vice versa. That is, if the services $S1$ and $S2$ interact by following the flow $op1 \rightarrow op2$, the choreography is fulfilled. Vice versa, if the services $S1$ and $S2$ interact by following the flow $op2 \rightarrow op1$, the choreography is not respected. That is, the interaction flow $op2 \rightarrow op1$ is an undesired interaction since, differently from what is specified by the choreography CLTS, the task $op2$ is performed before the task $op1$. As shown in the figure, a coordination delegate, $CD_{1.2}$, is automatically synthesized and interposed between $S1$ and $S2$ in order to prevent this interaction.

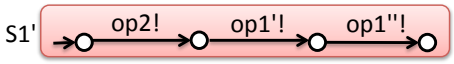

Fig. 3. $S1'$: a service discovered to play the role of $p1$

Let us suppose that, instead of discovering $S1$, another service, say $S1'$, would have been discovered. The interaction protocol of $S1'$ is shown in Fig. 3. Let us suppose also that the flow $op1'! \rightarrow op1''!$ (in $S1'$) is semantically equivalent to (yet syntactically different from) $op1!$ (in $S1$). Thus, in order to adapt the protocol of $S1'$ to the one of $p1$, we synthesize an adaptor that reorders the sequence of messages $op2! \rightarrow op1'! \rightarrow op1''!$ into $op1'! \rightarrow op1''! \rightarrow op2!$ and, then, merges the sequence of messages $op1'! \rightarrow op1''!$ into the single message $op1!$. This adaptor is synthesized as a wrapper for $S1'$ with a modular architecture resulting in the concurrent execution of two mediators, one for performing message reordering and the other one for performing the merge of messages. Note that the same adaptor could be used in the case we would consider a service that behaves exactly like $S1$ and its behaviour is changed afterwards so to become the one of $S1'$. This points out the ability of our modular adaptors to achieve resilient choreographies. That is choreographies able to evolve in response of possible changes in the participant services, while still keeping the prescribed coordination.

3 Choreography modular adaptors synthesis

As informally discussed in the previous section, a modular adaptor is automatically synthesized as a suitable composition of independent *mediators*. A mediator has an input-output behaviour (not necessarily strictly sequential, e.g., for allowing reordering of messages), and it is a "reactive" software entity harmonizing the interaction between heterogeneous services by intercepting output messages from one service and eventually issuing to another service (e.g., a CD) the *co-related* input messages. Message co-relations can be inferred by taking into account ontological information. In particular, we assume the existence of a *Domain Ontology* (DO) that can be used to semantically enrich the protocol description of the considered services. DO represents the relations holding between the various concepts used by the services to be mediated. Typically, ontologies account for two fundamental relations between concepts:

subsumption and *aggregation* [3]. A concept a is *subsumed by* a concept b, if the set denoted by a is a subset of the set denoted by b. A concept a is *an aggregate of* concepts $b_1, \ldots, b_n$ if the latter are part of the former. It is worth to mention that our use of the ontology concept is specific of the CHOReOS project. Thus, in the following, we will exploit these notions to our purposes. That is, concepts in DO correspond to service input/output operations. The two relations between concepts are, then, used to account for the granularity of the data that define the structure of the messages exchanged by the respective input/output actions. Indeed, in the current practice of ontology development, one cannot expect to find a highly specific (to the considered services) ontology as DO. The production of DO involves the extension of a more general ontology in the application domain. This extension allows the definition of specific ontologies that represent a semantic description for the considered services, respectively. Then DO results from discovering mappings between these ontologies. Note that nowadays there exist several ontologies (e.g., for e-commerce domains, see at: `http://www.heppnetz.de/projects/goodrelations/`) that can serve as common descriptions of specific domains, which can be shared among different applications. Furthermore, they are expressed by using languages (e.g., OWL, DAML, OIL, RDF Schema, just to mention a few) that allow ontology extension and automated reasoning for ontology mapping discovery [11].

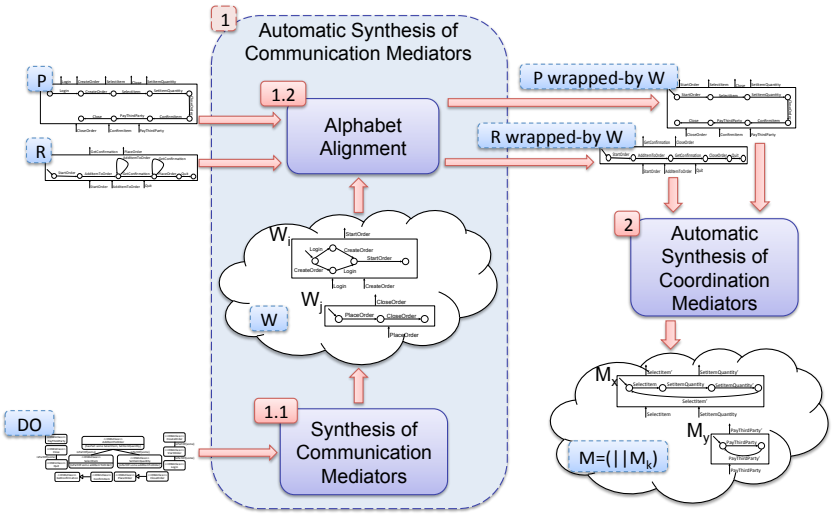

Fig. 4. Overview of the choreography modular adaptor synthesis

Our modular adaptor synthesis method is organized into two phases. In this paper we do not go into the details of the two phases that are rigorously described in [10], we rather give an overview of them. Fig. 4 pictorially shows the phases (as rounded-corner rectangles) with their related input/output artefacts. The numbers denote the order in

which the phases are carried out. The first phase splits into two sub-phases (1.1 and 1.2); it takes as input a domain ontology DO, for services (indeed, for service behavioural descriptions) P and R, and automatically synthesizes a set, W, of *Communication Mediators* (CMs). CMs are responsible for solving *communication mismatches*. They concern the semantics and granularity of the service protocol actions. To solve these kind of mismatches it is necessary to assume and use ontology knowledge in order to align the two protocols to the same concepts and language. In particular, the CMs in W are used as wrappers for P and R so to "align" their different alphabets to the same alphabet. Roughly speaking, the goal of this phase is to make two heterogeneous service protocols "speak" the same language. To this aim, the synthesized CMs translate an action from an alphabet into a certain sequence of actions from another alphabet (e.g., as illustrated in the previous section, through the merge of messages). However, despite the achieved alphabet alignment, *coordination mismatches* are still possible (e.g., as illustrated in the previous section, some message reordering is needed); the second phase is for solving such mismatches. *Coordination mismatches* concern the control structure of the protocols and can be solved by means of the mediator that can mediate the conversation between the two protocols so that they can actually interact. The synthesis of *COordination Mediators* (COMs) is carried out by reasoning on the traces of the "wrapped" P and R. As detailed in [10], for all pairs of traces, if possible, a COM that makes the two traces interoperable is synthesized. The parallel composition of the synthesized COMs represents, under alphabet alignment, the correct modular adaptor for P and R.

4 Explanatory example

In this section a simple and generic explanatory example is used to show the synthesis approach at work. This example should not be interpreted as a motivating one; rather it serves just to provide the reader with some more details about the single phases of our method.

By applying model transformation rules, the BPMN2 choreography diagram of Fig. 5 is transformed into the corresponding CLTS diagram in Fig. 6 (the CLTS diagram has been drawn by means of a graphical editor we have aptly developed).

For now, let us focus on the role of $p4$ only. Fig. 7 shows: (i) the interaction protocol expected for $p4$, (ii) the one of $S4$, i.e., a service discovered for playing the role of $p4$, and (iii) the assumed domain ontology (informally represented in the figure). By exploiting the ontology knowledge in Fig. 7, we can consider $S4$ as a suitable participant with respect to $p4$ since, although some of its operations are syntactically different from the ones of $p4$, they are still semantically co-related. Thus, under semantic co-relation of messages, $S4$ and $p4$ represent equivalent protocols except for the messages $op3$ and $op3'$ that are semantically different. As discussed in Section 3, in order to adapt $S4$ to $p4$ we synthesize communication and coordination mediators whose parallel composition represents the modular adaptor for $S4$. In particular, as shown in Fig. 8, the synthesized adaptor is made of three mediators; $M1$ and $M2$ are the communication

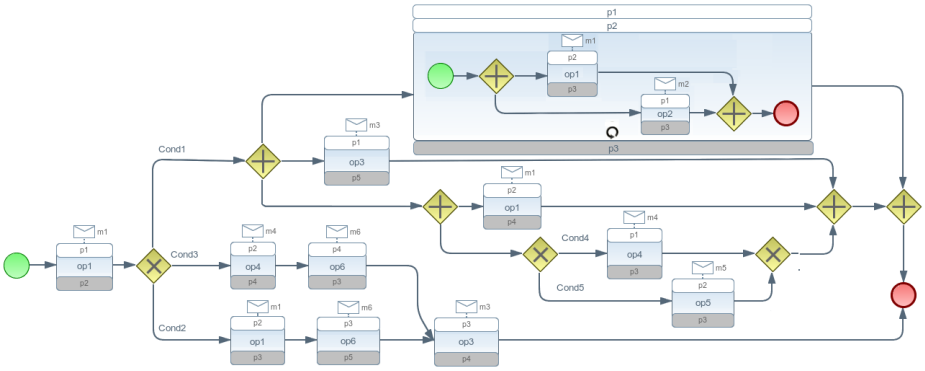

Fig. 5. BPMN2 choreography diagram example

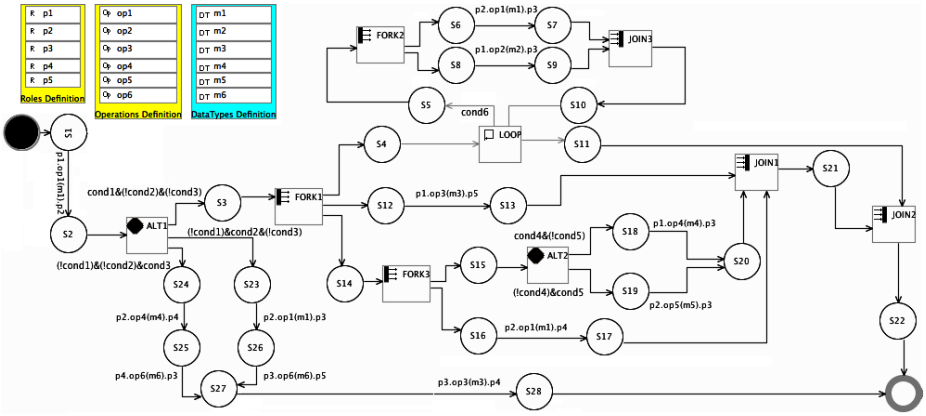

Fig. 6. CLTS derived from the BPMN2 choreography diagram in Fig. 5

mediators that perform the adaptation prescribed by the domain ontology, $M3$ is the coordination mediator that translates $op3$ into $op3'$.

Once we have adapted the protocol of the discovered services so to exactly match the one of their respective roles, our method can perform the synthesis of the needed CDs by reasoning abstractly on the roles' protocol. This is done by distributing the obtained CLTS into a set of coordination models. The latter contain coordination information codified as a set of tuples (called coordination tuples). For each interface that a participant p_i requires from another participant p_j, a coordination model $M_{CD_{pi.pj}}$ is derived. The model $M_{CD_{pi.pj}}$ will be then the input of the coordination delegate $CD_{pi.pj}$ that is interposed between the services acting as p_i and p_j.

For the convenience of the reader, before describing the format of the coordination tuples contained into the coordination models, Fig. 9 shows the set of CDs that are

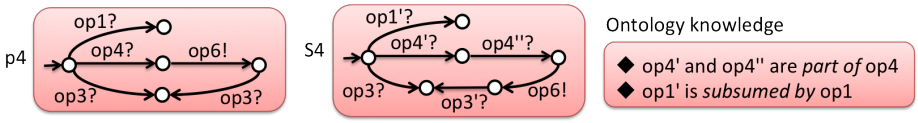

Fig. 7. Interaction protocol for the role of $p4$ and for $S4$, and the ontology knowledge

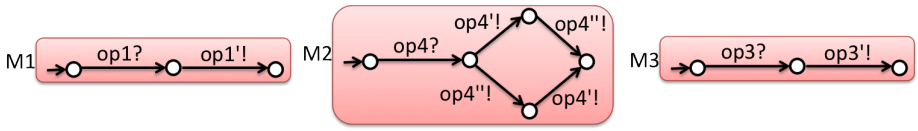

Fig. 8. The mediators constituting the modular adaptor for $S4$

generated out of the obtained CLTS and how they are interposed between the discovered (and adapted) services.

In Table 1 we provide a plain-text representation of some of the coordination tuples as contained in some of the coordination models derived for the example of Fig. 5. For space limitation we cannot show all the tuples for all the coordination models. Each tuple is composed of eight elements:

The **first element** denotes the CLTS source state from which the related CD can either perform the operation specified as **second element** of the tuple or take a move without performing any operation (i.e., the CD can step over an epsilon transition). In both cases, the **third element** denotes the reached target state. For instance, the first tuple of $M_{CD_{p1.p2}}$ specifies that the coordination delegate $CD_{p1.p2}$ can perform the operation $op1$ with message $m1$ from the source state $S1$ to the target state $S2$; whereas, the second tuple of $M_{CD_{p1.p2}}$ specifies that the coordination delegate $CD_{p1.p2}$ can step over the state $S2$ and reach the state $ALT1$, from where alternative branches can be undertaken. That is, as specified by the third, fourth and fifth tuple, the coordination delegate $CD_{p1.p2}$ can reach either the state $S3$, or $S23$, or $S24$, respectively, according to the evaluation of the related conditions.

The **fourth element** contains the set of states and related CDs that must be asked for to check whether the specified (allowed) operation can be forwarded or not. This

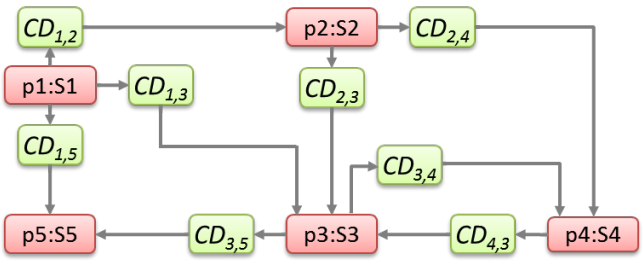

Fig. 9. Architecture of the example

$\mathbf{M_{CD_{p1.p2}}}$
$\langle S1, op1(m1), S2, Ask(), CD\{\}, true, Notify(), Wait()\rangle$ $\langle S2, \{\}, ALT1, Ask(), CD\{\}, true, Notify(), Wait()\rangle$ $\langle ALT1, \{\}, S3, Ask(), CD\{\}, cond1\&(!cond2)\&(!cond3), Notify(), Wait()\rangle$ $\langle ALT1, \{\}, S23, Ask(), CD\{2.3\}, (!cond1)\&cond2\&(!cond3), Notify(), Wait()\rangle$ $\langle ALT1, \{\}, S24, Ask(), CD\{2.4\}, (!cond1)\&(!cond2)\&cond3, Notify(), Wait()\rangle$ $\langle S3, \{\}, FORK1, Ask(), CD\{\}, true, Notify(), Wait()\rangle$ $\langle FORK1, \{\}, S14, Ask(), CD\{\}, true, Notify(), Wait()\rangle$ ...
$\mathbf{M_{CD_{p2.p3}}}$
$\langle S23, op1(m1), S26, Ask(), CD\{3.5\}, true, Notify(), Wait()\rangle$ $\langle S21, \{\}, JOIN2, Ask(), CD\{\}, true, Notify(S20\ to\ CD(2.3, 1.3, 1.5, 2.4)), Wait(S11\ from\ CD(2.3\ or\ 1.3), S13\ from\ CD(1.5), S17\ from\ CD(2.4))\rangle$ $\langle S7, \{\}, JOIN3, Ask(), CD\{\}, true, Notify(S7\ to\ CD(1.3)), Wait(S9\ from\ CD(1.3))\rangle$ $\langle LOOP, \{\}, S11, Ask(), CD\{\}, !cond6, Notify(), Wait()\rangle$ $\langle S11, \{\}, JOIN2, Ask(), CD\{\}, true, Notify(S11\ to\ CD(1.5, 1.3, 2.3, 2.4)), Wait(S13\ from\ CD(1.5), S20\ from\ CD(1.3\ or\ 2.3), S17\ from\ CD(2.4))\rangle$ ...
$\mathbf{M_{CD_{p1.p3}}}$
$\langle S9, \{\}, JOIN3, Ask(), CD\{\}, true, Notify(S9\ to\ CD(2.3)), Wait(S7\ from\ CD(2.3))\rangle$ $\langle S11, \{\}, JOIN2, Ask(), CD\{\}, true, Notify(S11\ to\ CD(1.5, 1.3, 2.3, 2.4)), Wait(S13\ from\ CD(1.5), S20\ from\ CD(1.3\ or\ 2.3), S17\ from\ CD(2.4))\rangle$
$\mathbf{M_{CD_{p3.p5}}}$
$\langle S26, op6(m6), S27, Ask(), CD\{3.4, 2.1\}, true, Notify(), Wait()\rangle$
$\mathbf{M_{CD_{p2.p1}}}$
$\langle S27, op3(m3), S29, Ask(CD(3.4)\ for\ S27), CD\{\}, true, Notify(), Wait()\rangle$ $\langle S29, \{\}, FinalState, Ask(), CD\{\}, true, Notify(), Wait()\rangle$
$\mathbf{M_{CD_{p3.p4}}}$
$\langle S27, op3(m3), S28, Ask(CD(2.1)\ for\ S27), CD\{\}, true, Notify(), Wait()\rangle$ $\langle S28, \{\}, FinalState, Ask(), CD\{\}, true, Notify(), Wait()\rangle$

Table 1. Coordination Models Tuples

means that race conditions can arise when, at a given execution point, more than one service wants to perform an operation but, according to the choreography specification, only one must be unconditionally elected. For instance, in the state $S27$, the coordination delegate $CD_{p2.p1}$ can be in a race condition with the coordination delegate $CD_{p3.p4}$ (and viceversa), whenever both $p2$ and $p3$ are ready to request the operation $op3$ with message $m3$ to $p1$ and $p4$, respectively. To solve this race condition, the tuple $\langle S27, op3(m3), S29, Ask(CD(3.4)\ \ for\ \ S27), CD\{\},\ true, Notify(), Wait()\rangle$ contained in $M_{CD_{p2.p1}}$ informs the coordination delegate $CD_{p2.p1}$ that before forwarding the operation $op3$, it must ask the permission to the coordination delegate $CD_{p3.p4}$ about the inquired state $S27$. Complementarily, the same applies for the tuple $\langle S27, op3(m3), S28, Ask(CD(2.1)\ \ for\ \ S27), CD\{\}, true,\ Notify(), Wait()\rangle$ contained in $M_{CD_{p3.p4}}$. As extensively discussed in [2], race conditions are solved by

applying a suitable extension of the seminal algorithm proposed in [13]. Thus, in this paper the resolution of race conditions is not further discussed.

The **fifth element** contains the set of (identifiers of) those CDs whose supervised services became active in the target state, i.e., the ones that will be allowed to require some operation from the target state. This information is used by the "currently active" CD(s) to inform the set of "to be activated" CDs (in the target state) about the changing global state. For instance, upon the operation $op1$ is requested from $p2$ to $p3$, the coordination delegate $CD_{p2.p3}$ uses the fifth element $CD\{3.5\}$ of the first tuple in $M_{CD_{p2.p3}}$ to inform the CD $CD_{p3.p5}$ about the new global state $S26$.

The **sixth element** reports the condition expression to be checked to select the correct tuple, and hence the correct flow(s) in the CLTS. For example, referring to the third tuple of $M_{CD_{p1.p2}}$, if the condition expression $cond1\&(!cond2)\&(!cond3)$ evaluates to true, then the coordination delegate $CD_{p1.p2}$ can step over the alternative state $ALT1$ and reach $S3$.

The **seventh element** contains the joining state that a CD, when reaching a join state, must notify to the other CDs in the parallel path(s) of the same originating fork. Complementarily, the **eight element** contains the joining state(s) that must be waited for. For example, considering the tuple $\langle S7, \{\}, JOIN3, Ask(), CD\{\}, true, Notify(S7 \quad to \quad CD(1.3)), Wait(S9 \quad from \quad CD(1.3))\rangle$ of $M_{CD_{p2.p3}}$, the coordination delegate $CD_{p2.p3}$ notifies the joining state $S7$ to the coordination delegate $CD_{p1.p3}$, and wait for the state $S9$ from $CD_{p1.p3}$. On the other hand, considering the tuple $\langle S9, \{\}, JOIN3, Ask(), CD\{\}, true, Notify(S9 \quad to \quad CD(2.3)), Wait(S7 \quad from \quad CD(2.3))\rangle$ of $M_{CD_{p1.p3}}$, the coordination delegate $CD_{p1.p3}$ notifies the joining state $S9$ to the coordination delegate $CD_{p2.p3}$, and wait for the state $S7$ from $CD_{p2.p3}$.

5 Related Work

The approach presented in this paper is related to a number of other approaches that have been considered in the literature. In Section 5.1, we discuss valuable work in the literature concerning coordinator (*service choreographer*) synthesis in the W3C (`http://www.w3.org/`) point of view of the SOA style. Then, in Section 5.2, we discuss other relevant works in the *Component-Based Software Engineering* domain that concern the synthesis of *protocol adaptors*.

5.1 Automated protocol coordinator synthesis

Many approaches have been proposed in the literature aiming at composing services by means of BPEL, WSCI, or WS-CDL choreographers [6,7,14,20,25]. The common idea underlying these approaches is to assume a high-level specification of the requirements that the choreography has to fulfill and a behavioral specification of the services participating in the choreography. From these two assumptions, by applying data and control-flow analysis, the BPEL, WSCI or WS-CDL description of a centralized choreographer specification is automatically derived. This description is derived in order to

satisfy the specified choreography requirements. In particular, in [25], the authors propose an approach to derive service implementations from a choreography specification. The authors of [9] and [23] present different approaches to semi-automatic services composition (based on abstract functional blocks) and semantic service descriptions, respectively. In [18], the authors propose an automatic approach to service composition exploiting AI planning algorithms. In [20] assume that some services are reused and propose an approach to exploit wrappers to make the reused services match the choreography.

Most of the previous approaches concern orchestration that is the most common approach to service composition. Conversely, our approach is one of the few in the literature that consider choreography as a means for automatically composing services in a fully distributed way. Despite the fact that the works described in [20,25] focus on choreography, they consider the problem of checking choreography realizability. It is a fundamentally different problem with respect to the one considered in this paper, i.e., discovery-based choreography realizability enforcement.

In [21], the authors show how to monitor safety properties locally specified (to each component). They observe the system behavior simply raising a *warning message* when a violation of the specified property is detected. Our approach goes beyond simply detecting properties (e.g., a choreography specification) by also allowing their enforcement. In [21] the best thing that they can do is to reason about the global state that each component *is aware of*. Note that, differently from what is done in our approach, such a global state might not be the actual current one and, hence, the property could be considered guaranteed in an *"expired"* state. Another work in the area of the synthesis of runtime monitors from automata is described in [22]. Note that runtime monitoring is mostly focused on the detection of undesired behaviours, while runtime enforcement focuses on their prevention/solution.

5.2 Automated protocol adaptor synthesis

The mediation/adaptation of protocols have received attention since the early days of networking. Indeed many efforts have been done in several directions including for example formal approaches to protocol conversion, like in [8,12].

The seminal work in [29] is strictly related to the notions of mediator presented in this paper. Compared to our adaptor synthesis, this work does not allow to deal with ordering mismatches and different granularity of the languages (solvable by the split and merge primitives).

Recently, with the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [28,27]. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

In [24] the authors present an approach for formally specifying adaptor wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches. Although this formalizations supports modularization, automated synthesis is not treated at all hence keeping the focus only on adaptor design and specification.

In [19], the authors use a game theoretic approach for checking whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements. This approach is able to automatically synthesize the converter. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adaptor, by delegating to the user the non-trivial task of specifying that.

6 Conclusions and Future Work

In this paper, we proposed an automatic approach to enforce choreography realizability. The described methodology allows for both *adapting* the external interaction of the considered services to the roles of the choreography and (ii) *coordinating* the (adapted) interaction so to fulfill the global collaboration prescribed by the choreography.

To this end, the proposed approach uses model transformations to extract from a BPMN2 choreography specification the global coordination logic and codifies it into an extended LTS, called *Choreography LTS* (CLTS). The CLTS is then modularly distributed into a set of *adaptors* and *coordination delegates* that, when combined together, allow for enforcing the choreography in a fully distributed way, while adapting the services' interaction. The expressiveness of the CLTS model allows us to fully automate the approach and to transform very complex choreography specifications into powerful coordination and adaptation logics.

In Section 5, we related our approach to existing centralized solutions. Summing up, the most relevant advantage of our approach with respect to these solutions is that the degree of parallelism of the system is maintained despite the introduction of the adaptors and coordination delegates. Often, centralized approaches do not permit full parallelism since the adaptor/coordinator is usually implemented as a centralized single-threaded component and the communication with it is synchronous.

The proposed approach has already been applied to a large-scale realistic case study, namely the passenger-friendly airport scenario and a public demo is available at the CHOReOS web-site `http://www.choreos.eu`. Currently, we are applying the process at two other industrial case studies of CHOReOS in the domains of marketing and sales, and Internet of things. The results will also be publicly available by the CHOReOS web site. The current implementation of the whole approach supports the generation of Java code for coordinating SOAP-based Web-services. Considering the general-purpose nature of the approach, other languages and application domains are eligible, and other forms of wrapping can be easily realized.

The current approach allows supervised services to perform an operation that is outside the scope of the specified choreography. In this sense our approach is permissive, and can be parameterized to be either permissive or restrictive with respect to these operations. However, simply enabling or disabling the execution of operations outside the scope of the choreography is a trivial strategy. In the future we plan to investigate, and embed into the approach implementation, more accurate strategies to suitably deal with these operations.

A further interesting future direction is the investigation of non-functional properties of the choreography, e.g., by extending the choreography specification with performance or reliability attributes and accounting for them in the CDs synthesis process.

References

1. F. Arbab and F. Santini. Preference and similarity-based behavioral discovery of services. In *Web Services and Formal Methods*, volume 7843 of *Lecture Notes in Computer Science*, pages 118–133. 2013.
2. M. Autili, D. Ruscio, A. Salle, P. Inverardi, and M. Tivoli. A model-based synthesis process for choreography realizability enforcement. In *FASE*, volume 7793 of *LNCS*, pages 37–52. 2013.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.
4. S. Basu and T. Bultan. Choreography conformance via synchronizability. In *Proc. of WWW '11*, pages 795–804, 2011.
5. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL, pages 191–202. ACM, 2012.
6. A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC'06, volume 4294 of LNCS*, 2006.
7. D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi. Automatic service composition and synthesis: the roman model. *IEEE Data Eng. Bull.*, 31(3):18–22, 2008.
8. K. L. Calvert and S. S. Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas in Communications*, 8(1), 1990.
9. M. Fluegge and D. Tourtchaninova. Ontology-derived activity components for composing travel web services. In *International Workshop on Semantic Web Technologies in Electronic Business (SWEB2004)*, 2004.
10. P. Inverardi and M. Tivoli. Automatic synthesis of modular connectors via composition of protocol mediation patterns. In *ICSE*, pages 3–12, 2013.
11. Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowl. Eng. Rev.*, 18(1), 2003.
12. S. S. Lam. Correction to "protocol conversion". *IEEE Trans. Software Eng.*, 14(9), 1988.
13. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21:558–565, July 1978.
14. A. Marconi, M. Pistore, and P. Traverso. Automated Composition of Web Services: the ASTRO Approach. *IEEE Data Eng. Bull.*, 31(3):23–26, 2008.
15. OMG. Business Process Model And Notation (BPMN) Version 2.0. `http://www.omg.org/spec/BPMN/2.0/`.
16. M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.
17. P. Poizat and G. Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC 2012*, pages 1927–1934, 2012.
18. S. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *In Proc. of the 11th WWW Conference*, 2002.
19. Roberto Passerone and Luca De Alfaro and Thomas A. Henzinger and Alberto L. Sangiovanni-Vincentelli. Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin. In *ICCAD*, 2002.

20. G. Salaün. Generation of service wrapper protocols from choreography specifications. In *Proc. of SEFM*, 2008.
21. K. Sen, A. Vardhan, G. Agha, and G. Rosu. Efficient decentralized monitoring of safety in distributed systems. In *Proc. of ICSE*, 2004.
22. J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse. Runtime monitoring of web service conversations. *IEEE T. Services Computing*, 2(3), 2009.
23. E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *In Proc. of Web Services: Modeling, Architecture and Infrastructure workshop*, 2003.
24. B. Spitznagel and D. Garlan. A compositional formalization of connector wrappers. In *ICSE*, 2003.
25. J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *WS-FM*, pages 1–16, 2007.
26. E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A semantic approach to approximate service retrieval. *ACM Trans. Internet Technol.*, 8(1), 2007.
27. R. Vaculín, R. Neruda, and K. P. Sycara. An agent for asymmetric process mediation in open environments. In *SOCASE*, 2008.
28. R. Vaculín and K. Sycara. Towards automatic mediation of OWL-S process models. *Web Services, IEEE International Conference on*, 2007.
29. D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19, March 1997.