CHOReOSynt: Enforcing Choreography Realizability in the Future Internet[*]

Marco Autili, Davide Di Ruscio, Amleto Di Salle, Alexander Perucci
University of L'Aquila, Italy
{marco.autili|davide.diruscio|amleto.disalle|alexander.perucci}@univaq.it

ABSTRACT

Choreographies are an emergent Service Engineering (SE) approach to compose together and coordinate services in a distributed way. A choreography formalizes the way business participants coordinate their interactions. The focus is not on orchestrations of the work performed within them, but rather on the exchange of messages between these participants. The problems usually addressed when considering a choreography-based specification of the system to be realized are *realizability check*, and *conformance check*.

In this paper we describe the `CHOReOSynt` tool, which has been conceived to deal with an additional problem, namely, *automated choreography enforcement*. That is, when the goal is to actually realize a service choreography by *reusing third-party services*, their uncontrolled (or wrongly coordinated) composite behavior may show undesired interactions that preclude the choreography realization. `CHOReOSynt` solves this problem by automatically synthesizing additional software entities that, when interposed among the services, allow for preventing undesired interactions.

Screencast: `http://choreos.disim.univaq.it/downloads/`

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Miscellaneous

General Terms

Design, Languages

Keywords

Choreography Synthesis, Distributed Coordination

1. INTRODUCTION

Service-Oriented Computing (SOC)is now largely accepted as a well-founded reference paradigm for the Future Internet [7, 9] computing. The near future in service-oriented system development envisions an ultra large number of diverse service providers and consumers that can be composed to fit users' needs.The trend is to build modern applications by reusing and assembling distributed services rather than realizing stand-alone and monolithic programs [14, 11].

Today service composition is heavily based on Business Process Model, and mainly on service orchestrations that can be specified in various formats, such as BPEL [13] or BPMN 2.0 [12]. Service orchestration is a centralized approach to composing multiple services into a larger application. It works well in static environments where services are predefined and environment changes are minimal. This is a wrong assumption for the Future Internet, which envisions an ultra large number of diverse service providers and consumers that are impossible to coordinate using centralized approaches [9, 11].

Choreographies are an emergent Service Engineering (SE) approach to compose together and coordinate services in a distributed way. Service choreographies will certainly have an important role in shaping the SOC within the vision of Future Internet. A choreography formalizes the way business participants coordinate their interactions. The focus is not on orchestrations of the work performed within them, but rather on the exchange of messages between these participants. In this respect, a choreography defines the global expected behavior between interacting participants. More precisely, service choreography is a decentralized approach, which provides a looser way to design service composition by specifying tasks, their participants and message protocol between them but not coupling choreography tasks to concrete services. The need for service choreography was recognized also in BPMN 2.0, which introduced choreography modeling constructs. However, until now service choreographies were solely used for design purposes only as there was no technology support for enabling a smooth transition from service choreography design to execution.

Within the Future Internet, we imagine the establishment of service federations in which different providers and users interact, following specific rules established to foster and simplify inter-organization integration according to the requirements coming from a specific context. In this vision, service choreographies will contribute to the SOC vision within the Future Internet only if automatically supported by suitable design and development environments, as well as a set of infrastructural services.

The problems usually addressed when considering a choreography-based specification C of the system to be realized (where *roles* characterize the *expected* behavior of the participants) are: (a) *realizability check*, i.e., whether C can

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

be realized by implementing each participant as specified by the role to be played; and (b) *conformance check*, i.e., whether the global interaction of a set of services, discovered as suitable participants, satisfies C or not. In the research literature many approaches have been proposed to address these problems (e.g., [8, 14, 6, 5]). However, when the goal is to actually realize a service choreography by **reusing third-party services**, hence going beyond just checking its effectiveness, a further problem worth to be considered concerns automated choreography *enforcement*. That is, how to coordinate the interactions of a set of services discovered as suitable participants in order to fit the choreography specification. This requires to distribute and enforce, among the participants, the global coordination logic extracted from the choreography specification.

The general problem here is that, although for each choreography participant a suitable third-party service may have been discovered (and hence, its interaction behavior fits the behavior of the participant in isolation), the uncontrolled (or wrongly coordinated) composite behavior of all the discovered services may show undesired interactions that prevent the choreography realization. For a detailed and formal description of the notion of undesired interaction, please refer to [1, 3, 4].

In this paper we present `CHOReOSynt`, a tool that has been developed in the context of the EU project CHOReOS (`http://www.choreos.eu`). `CHOReOSynt` leverages model-based methodologies and relevant SOA standards in order to make choreography development a systematic process to the reuse and the assembling of services discovered within the Future Internet. `CHOReOSynt` enables the construction of choreography-based service-oriented systems by introducing a model-based development process and associated synthesis methods for choreographing services. A detailed description of the underlying methodology implemented by `CHOReOSynt` can be found in [1, 2, 3, 4, 10]. `CHOReOSynt` has been released also under the FISSi initiative (`http://www.ow2.org/view/Future_Internet`).

2. BACKGROUND

The CHOReOS project concerns the development of choreography-based service-oriented systems. To this aim, within CHOReOS we have precisely defined a development process and implemented a supporting Integrated Development and Run-time Environment (IDRE). At a very high-level, the development phase consists of a set of *activities* (see Figure 1), and related artifacts, that allow to:

A.1. specify requirements by using a dedicated tool,
A.2. derive an initial choreography specification from them, given in BPMN 2.0,
A.3. refine the initial choreography specification to allow for automation,
A.4. concretize the choreography specification by discovering services suitable to play the roles of the choreography,
A.5. assess the quality of the choreography, and
A.6. automatically synthesize a distributed choreographer to be then used to enact the choreography, hence enforcing the choreography realizability.

According to the described CHOReOS process, we proposes a model-driven synthesis approach to generate from a

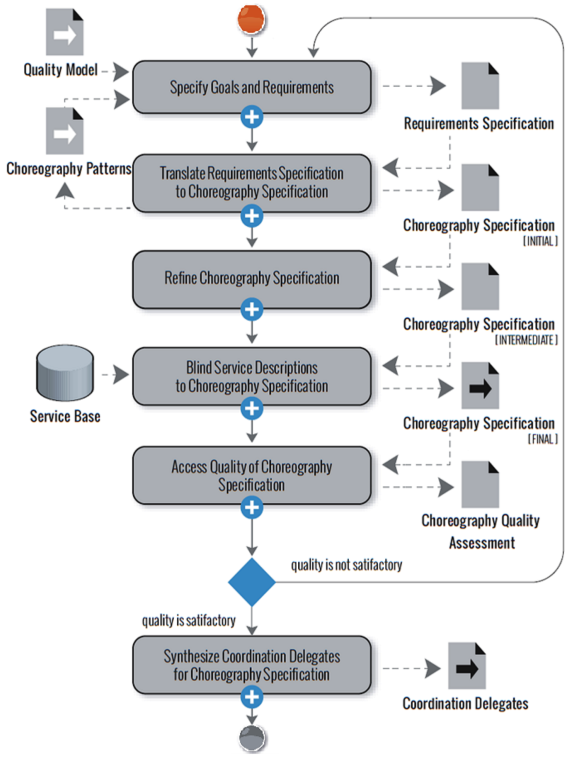
Quality Model
Specify Goals and Requirements
Requirements Specification
Choreography Patterns
Translate Requirements Specification to Choreography Specification
Choreography Specification [INITIAL]
Refine Choreography Specification
Choreography Specification [INTERMEDIATE]
Service Base
Blind Service Descriptions to Choreography Specification
Choreography Specification [FINAL]
Access Quality of Choreography Specification
Choreography Quality Assessment
quality is not satifactory
quality is satifactory
Synthesize Coordination Delegates for Choreography Specification
Coordination Delegates

Figure 1: Development Process

BPMN 2.0 choreography model C an automata-based specification (called CLTS) of the coordination logic "implied" by the choreography. A CLTS provides an explicit description of the allowed sequences of In/Out service operation calls and, as such, represents an intermediate representation of the choreography that allows for the extraction of the complex coordination logics "hidden" in BPMN 2.0 choreography specifications [1, 4, 10]. Basically, a CLTS is an extended Labeled Transition System (LTS) that allows handling complex constructs of BPMN 2.0 Choreography Diagrams, such as gateways, loops, forks and joins. 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. The synthesis processor automatically derives these software entities, called Coordination Delegates (CDs), and interposes them among the participant services according to the CHOReOS architectural style (see Figure 2). The generated CDs are then able to properly access and coordinate the discovered services relying on the communication facilities provided by the CHOReOS middleware.

More precisely, the coordination logic modelled by the derived CLTS is distributed into a set of *Coordination Models* (CMs), one for each CD. The CDs exploit the CMs in order to coordinate the interaction of the participants in a way that the resulting collaboration realizes the choreography specified by C. CDs perform pure business-level coordination by intercepting/proxifying the service interaction (i.e., standard communication in Figure 2), and mediate it by exchanging the coordination information (i.e., additional communication in the figure) contained in their CMs. In this way, the CDs prevent possible *undesired interactions*. The latter are those interactions that do not belong to the set of interactions allowed by C and can happen when the discovered third-party services collaborate in an uncontrolled way. CDs implement a distributed coordination algorithm that we have formally defined to describes the coordination logic to be performed by a CD, while relying on its CM.

3. CHOREOSYNT

In this section we outline the `CHOReOSynt` tool, which supports the activity *A.6* mentioned in the previous section. `CHOReOSynt` consists of a number of RESTful services, and a set of Eclipse plugins that have been developed to interact with such services. By referring to Figure 3, the `Choreography Synthesis Processor` allows for deriving the CD artefacts from the BPMN 2.0 specification of the choreography. To this end, model transformations are employed and interoperation with the `Service Discovery` is required (not in the focus of this paper). The coordination delegates, when deployed by the `Enactment Engine` (not in the focus of this paper), allow for enacting the choreography by realizing the distributed coordination logic between the discovered services. The services offer bespoke functionalities to:

- start the synthesis process giving as input a BPMN 2.0 Choreography Diagram;
- transform the BPMN2 Choreography Diagram into an extended LTS called choreography LTS (CLTS);
- derive a set of Coordination Models containing information that serve to coordinate the services involved in the choreography in a distributed way;
- extract the participants of the choreography and project the choreography on their behavioral role;
- simulate the behavioral role of the participants in the choreography against the behavior of the services discovered by the eXtensible Service Discovery;
- generate the Coordination Delegate artefacts and the so called "`ChorSpec`" specification to be used by the Enactment Engine component for deploying and enacting the choreography;

Such functionalities are implemented by the components of the `Choreography Synthesis Processor` shown in Figure 3 and described below.

M2M Transformator – The Model-to-Model (M2M) Transformator offers a set of model transformations. Specifically, it offers a operation `bpmn2clts()` that takes as input the BPMN specification of the choreography and performs the model-to-model transformation to derive the CLTS. The transformation is implemented by means of the Atlas Transformation Language (ATL - `http://www.eclipse.org/atl/`) which is a domain specific language for realizing model-to-model transformations.

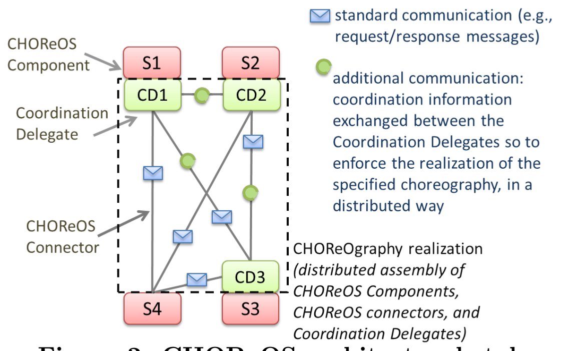
standard communication (e.g., request/response messages)
CHOReOS Component
S1
S2
CD1
CD2
additional communication: coordination information exchanged between the Coordination Delegates so to enforce the realization of the specified choreography, in a distributed way
Coordination Delegate
CHOReOS Connector
CD3
CHOReOgraphy realization (distributed assembly of CHOReOS Components, CHOReOS connectors, and Coordination Delegates)
S4
S3

Figure 2: CHOReOS architectural style

Then, starting from the CLTS specification of the choreography, the synthesis process extracts the list of the participants and, applying a further model-to-model transformation, automatically derives, for each participant, the CLTS model of the expected behavior with respect to the specified choreography. To this end another operation named `extractParticipants()` is offered. The CLTS model of expected behavior is achieved by projecting (`projection()`) the choreography onto the participant, hence filtering out those transitions, and related states, that do not belong to the participant. Basically, for each participant, this CLTS model specifies the interaction behavior that a candidate service (to be discovered) has to support in order to be able to play the role of the participant in the choreography.

Synthesis Discovery Manager – The Synthesis process and the Discovery process interact each other to retrieve, from the service base, those candidate services that are suitable for playing the participant roles required by the choreography specification, and hence, those services whose (offered and required) operations and behavior are compatible with the expected behavior as extracted from the choreography through projection. In particular, for each participant, the Synthesis Processor interacts with the Synthesis Discovery Manager by performing the call to the `discoverServices()` operation that takes the participant (abstract) CLTS as input. Then, a query is issued to the eXtensible Service Discovery (XSD) component.

Behavior Simulator – Once a set of concrete candidate services has been discovered, the synthesis process has to select them by checking, for each participant, if its expected behavior can be simulated by some candidate service. Note that, for a given participant, behavioral simulation is required since, although the discovered candidate services for it are able to offer and require (at least) the operations needed to play the role of the participant, one cannot be sure that the candidate services are able to support the operations flow as expected by the choreography. Thus, in order to simulate the expected behavior of a participant with the behavior of a service, the Behavior Simulator offers a operation named `simulation()` that takes as input the projected (abstract) CLTS of the participant and the extended (concrete) LTS of the service as retrieved by the URI returned by the discovery service. It might be interesting to mention that the simulation method implements a notion of strong simulation suitably extended to treat the CLTSs and extended the LTSs we use in CHOReOS. After simulation, if all the participant roles have been "covered" by (some of) the discovered services, the abstract CLTS is concretized with the actual names of the selected services and the actual names of the offered and requested operations. This step is performed by the operation `absctractCLTS2concreteCLTS()`. Then, the automated synthesis process distributes the coordination logic specified by the obtained CLTS into a set of Coordination Models by means of the functionality `clts2coord()`.

Coordination Delegate Generator – Once the services have been selected for all the choreography participants, and hence the CLTS has been concretized, the synthesis processor can generate the Coordination Delegates through the operation `generateCD()` offered by the Coordination Delegate Generator component.

Next step in the process – Once the Coordination

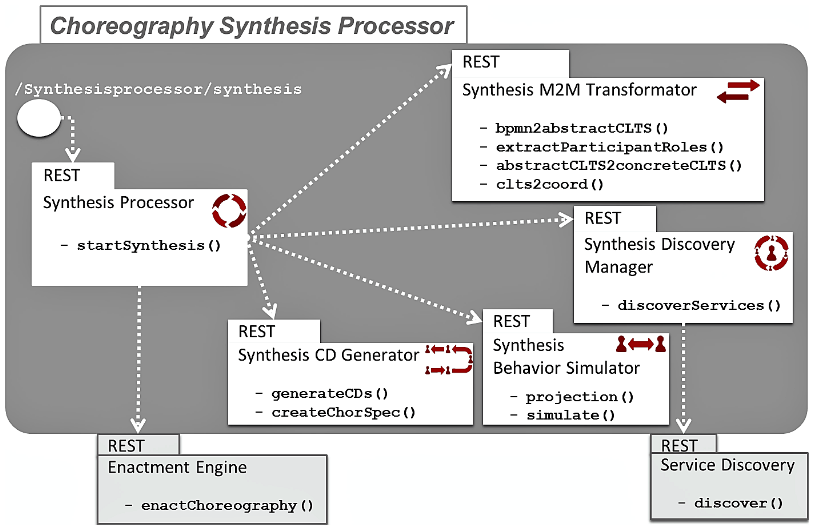

Figure 3: REST Architecture of the Synthesis Processor

Delegates have been generated, the Coordination Delegate Generator component can further generate a specification of the choreography (called `ChorSpec`) to be passed to the Enactment Engine. To this end, the operation `createChorSpec()` is offered. It takes as input the selected services and the coordination delegates generated for them. The `ChorSpec` is an XML-based declarative description of the choreography that specifies the locations of the selected services and of the generated Coordination Delegate artifacts that can be deployed. Indeed, before passing the `ChorSpec` to the Enactment Engine, the Choreography Offline Testing process activity is performed to assess the quality of the choreography specification, its well formedness, etc.

4. CONCLUSIONS

In this paper we described `CHOReOSynt`, a tool which has been conceived in the context of the EU CHOReOS project to deal with the *automated choreography enforcement* problem when developing software systems in terms of choreographies of services. The tool consists of a number of RESTful services, which have been integrated into the CHOReOS Integrated Development and Run-time Environment (IDRE).

`CHOReOSynt` has been successfully applied to a number of use cases in different applicative domains, e.g., dynamic marketing planning and strategies, air transportation, and for managing fleets of taxis in large cities. The use cases have highlighted the benefits of using the synthesis approach to automatically realize choreographies, and to coordinate (in a distributed way) complex interactions involving a large number of services. To download `CHOReOSynt` and for more details about its evaluation interested readers can refer to `http://choreos.disim.univaq.it/` and `http://choreos.eu/bin/view/Develop/Forge`.

5. REFERENCES

[1] M. Autili, D. Di Ruscio, A. Di 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.

[2] M. Autili, D. Di Ruscio, P. Inverardi, James Lockerbie, and M. Tivoli. A development process for requirements based service choreography. In *RESS*, pages 59–62, 2011.

[3] M. Autili, A. Salle, and M. Tivoli. Synthesis of resilient choreographies. In *SERENE*, volume 8166 of *LNCS*, pages 94–108. 2013.

[4] M. Autili and M. Tivoli. Distributed enforcement of service choreographies. In *FOCLASA*, 2014.

[5] S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *Procs. of POPL*, pages 191–202. ACM, 2012.

[6] Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In *Proc. of WWW '11*, pages 795–804, 2011.

[7] Bruxelles: European Commission. Future Internet 2020: Visions of an Industry Expert Group, 2009.

[8] D. Calvanese, G. De 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.

[9] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M.A. Gerosa, and A. Hamida. Service-oriented middleware for the future internet: state of the art and research directions. *JISA Journal*, 2(1):23–45, 2011.

[10] A. Di Salle M. Autili, D. Di Ruscio and P. Inverardi. Synthesizing an automata-based representation of bpmn2 choreography diagrams. In *ModComp at MoDELS'14*, 2014.

[11] G. De Angelis M. Autili, A. Ben Hamida and D. Silingas. *Composing Services in the Future Internet: Choreography-Based Approach*, volume Intelligent BPM Systems (iBPMS) Book: Impact and Opportunity, pages 163–174. 2013.

[12] Object Management Group (OMG). Business Process Model and Notation (BPMN), version 2.0, 2011.

[13] Organization for the Advancement of Structured Information Standards (OASIS). BPEL 2.0, 2007.

[14] Pascal Poizat and Gwen Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC 2012*, pages 1927–1934, 2012.