CHOReVOLUTION: Hands-On In-Service Training for Choreography-Based Systems

Marco Autili, Amleto Di Salle, Claudio Pompilio(✉), and Massimo Tivoli

University of L'Aquila, L'Aquila, Italy
{marco.autili,amleto.disalle,claudio.pompilio,massimo.tivoli}@univaq.it

Abstract. CHOReVOLUTION is a platform for the tool-assisted development and execution of scalable applications that leverage the distributed collaboration of services specified through service choreographies. It offers an Integrated Development and Runtime Environment (IDRE) comprising a wizard-aided development environment, a system monitoring console, and a back-end for managing the deployment and execution of the system on the cloud. In this tutorial paper, we describe the platform and demonstrate its step-by-step application to an industrial use case in the domain of Smart Mobility & Tourism.
(Demo Video: youtu.be/ae2jI9SYsvg)
(GitHub: https://github.com/chorevolution/CHOReVOLUTION-IDRE)

Keywords: Service choreographies · Automated synthesis · Distributed computing · Distributed coordination · Adaptation

1 Introduction

The Future Internet [15] is now a reality that reflects the changing scale of the Internet. The expanding network infrastructure is supporting the today's trend toward the fruitful cooperation of different business domains through the interorganizational composition of a virtually infinite number of software services[1]. This vision is embodied by reuse-based service-oriented systems, in which services play a central role as effective means to achieve interoperability among different parties of a business process, and new systems can be built by reusing and composing existing services.

Service choreographies are a form of decentralized composition that model the external interaction of the participant services by specifying peer-to-peer message exchanges from a global perspective. When third-party (possibly black-box) services are to be composed, obtaining the distributed coordination logic

[1] http://www.fiware4industry.com.

Published by Springer Nature Switzerland AG 2020
S. Bliudze and L. Bocchi (Eds.): COORDINATION 2020, LNCS 12134, pp. 3–19, 2020.
https://doi.org/10.1007/978-3-030-50029-0_1

required to enforce the realizability of the specified choreography is non-trivial and error prone. Automatic support is then needed [1,3].

The CHOReVOLUTION H2020 EU project[2] develops a platform for the generation and execution of scalable distributed applications that leverage the distributed collaboration of services and things by means of service choreographies. In particular, it realizes an Integrated Development and Runtime Environment (IDRE) that comprises a wizard-aided development environment, a system monitoring console, and a back-end for managing the deployment and execution of the system on the cloud.

The CHOReVOLUTION IDRE makes the realization of choreography-based smart applications easier by sparing developers from writing code that goes beyond the realization of the internal business logic related to the provisioning of the single system functionalities, as taken in isolation. That is, the distributed coordination logic, which is needed to realize the global collaboration prescribed by the choreography specification, is automatically synthesized by the IDRE, without requiring any specific attention by developers for what concerns coordination aspects. Furthermore, developers can also more easily reuse existing consumers/providers services. These aspects have been appreciated by the industrial partners in that the approach permits to develop distributed applications according to their daily development practices.

The IDRE is open-source and free software, available under Apache license. The binaries and the source code of version 2.2.0 can be downloaded at the following URL https://github.com/chorevolution/CHOReVOLUTION-IDRE/releases. Documentation[3] is also available.

The paper is organized as follows. Section 2 briefly introduces the problem solved by the CHOReVOLUTION IDRE together with a brief discussion on related work. Section 3 describes the overall approach supported by CHOReVOLUTION, and Sect. 4 describes the actual development process supported by IDRE. Section 5 gives an overview of the main components constituting the IDRE. Section 6 presents the IDRE at work on an industrial use case in the Smart Mobility and Tourism domain, and Sect. 7 concludes the paper.

2 Problem Statement and Related Works

Choreographies model peer-to-peer communication by defining a multiparty protocol that, when put in place by the cooperating participants, allows reaching the overall choreography goal in a fully distributed way. In this sense, choreographies differ significantly from other forms of service composition such as orchestrations, where all participants (but the orchestrator) play the passive role of receiving requests by the orchestrator only.

So far, choreographies have been solely used for design purposes, simply because there was no technological support for enabling a smooth transition from choreography design to execution. In the literature, many approaches have

[2] http://www.chorevolution.eu.

[3] https://github.com/chorevolution/CHOReVOLUTION-IDRE/wiki/User-Guide.

been proposed to deal with the foundational problems of checking choreography realizability, analyzing repairability of the choreography specification, verifying conformance, and enforcing realizability [4,8,10,11,14,18,19]. These approaches provide researchers with formal means to address fundamental aspects of choreographies. They are based on different interpretations of the choreography interaction semantics, concerning both the subset of considered choreography constructs, and the used formal notations.

The need for practical approaches to the realization of choreographies was recognized in the OMG's BPMN 2.0[4] standard, which introduces dedicated *Choreography Diagrams*, a practical notation for specifying choreographies that, following the pioneering BPMN process and collaboration diagrams, is amenable to be automatically treated and transformed into actual code. BPMN2 choreography diagrams focus on specifying the message exchanges among the participants from a global point of view. A participant role models the expected behavior (i.e., the expected interaction protocol) that a concrete service should be able to perform to play the considered role.

When considering choreography-based systems, the following two problems are usually taken into account: (i) *realizability check* – checks whether the choreography can be realized by implementing each participant so as 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., [8,9,11,13,16,17,19–21].

However, to put choreographies into practice, we must consider realizing them by reusing third-party services. This leads to a further problem: the *automatic realizability enforcement* problem. It can be informally phrased as follows.

> Problem statement: *given a choreography specification and a set of existing services, externally* ***coordinate*** *and* ***adapt*** *their interaction so to fulfill the collaboration prescribed by the choreography specification.*

By taking as input a BPMN2 Choreography Diagram, and by exploiting a service inventory where existing services are published in order to be reused for choreography realization purposes, a set of software artefacts are automatically generated in order to implement the adaptation and distributed coordination logic prescribed by the choreography specification. These artefacts adapt and coordinate the interaction among the services – selected as suitable choreography participants – in order to ensure that their distributed cooperation runs by performing the flows specified in the BPMN2 Choreography Diagram only, hence preventing both interface and interoperability mismatches (application- and middleware-level adaptation) and the execution of possible flows violating the specification (correct coordination). Furthermore, when needed, specific security policies can be enforced on the participants interaction so to make the choreography secure. These policies concern correct inter-process authentication and authorization. The generated artefacts are:

[4] http://www.omg.org/spec/BPMN/2.0.2/.

- **Binding Components (BCs)** serve to ensure middleware-level interoperability among the possibly heterogenous services involved in the choreography. For instance, a BC can be generated in order to make a SOAP web service able to communicate with a REST service.
- When needed, **Security Filters (SFs)** secure the communication among involved services by enforcing specified security policies.
- Abstract services defined in the choreography specification characterizes the expected interface of the choreography participants. When using the IDRE to implement the specified choreography participants, concrete services (possibly black-box) are selected from a service inventory and reused. Thus, a concrete service has to match the interface of the participants it has to realize. Here, **Adapters (As)** come into place. That is, if needed, an Adapter is used to adapt the interface of a concrete service in order to match the one of the abstract service it implements.
- **Coordination Delegates (CDs)** supervise the interaction among the involved participants in order to enforce the service coordination logic prescribed by the choreography specification in a fully-distributed way. In other words, CDs act as distributed controllers. That is, they ensure that the distributed interaction among the reused concrete services will run according to the execution flows described by the choreography specification, hence preventing distributed interactions that could violate the specification.

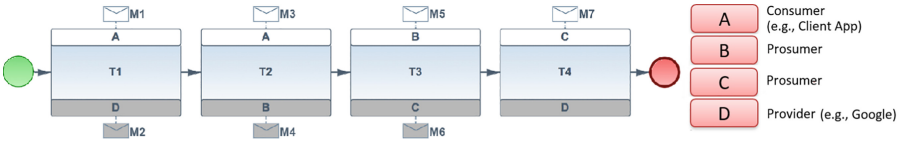

Fig. 1. BPMN2 choreography diagram example

For those readers new to choreographies, Fig. 1 shows a simple example of a BPMN2 Choreography Diagram. Choreography diagrams define the way business participants coordinate their interactions. The focus is on the exchange of messages among the involved participants. A choreography diagram models a distributed process specifying activity flows where each activity represents a message exchange between two participants. Graphically, a choreography task is denoted by a rounded-corner box. The two bands, one at the top and one at the bottom, represent the participants involved in the interaction captured by the task. A white band is used for the participant initiating the task that sends the initiating message to the receiving participant in the dark band that can optionally send back the return message.

The choreography in Fig. 1 involves four participants, `A`, `B`, `C`, and `D`, for the execution of four sequential tasks, `T1`, `T2`, `T3` and `T4`. Specifically, `A` sends the message `M1` to `D`, enabling it for the execution of `T1`. After that, `D` replies to `A` by sending the message `M2`. At this point, `A` sends `M3` to `B` that, after the execution of `T2`, replies `M4` to `A` and sends `M5` to `C`. Only when `M5` is received by `C`, it executes `T3`, replies `M6` to `B` and sends `M7` to `D`. Finally, `D` executes `T4` and the choreography ends.

By analyzing the choreography, we can distinguish three different types of participants: *consumer*, *provider*, and *prosumer* (i.e., both consumer and provider). For instance, considering a reuse-based development scenario in which existing services are published in a suitable service inventory, the consumer participant `A` might be played by an existing Client App; the provider participant `D` by an existing Web Service, e.g., Google Maps; `B` and `C` might be two prosumers that have to be developed from scratch in order to realize the choreography.

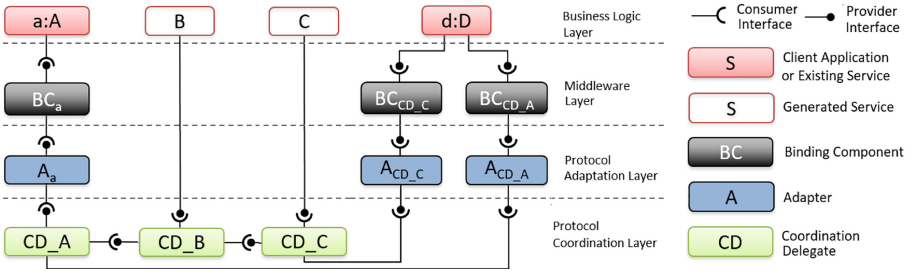

Fig. 2. Choreography architectural style (a sample instance of)

Figure 2 shows architecture of the system that realizes the choreography specified in Fig. 1, and that is automatically generated by the IDRE. The top-most layer contains the services representing the business logic. In particular, `a:A` denotes that the role of the consumer participant `A` is played by `a`, the Client App in our example; `d:D` denotes that the role of the provider participant `D` is played by `d`, an existing provider service to be reused, whereas, concerning the participants `B` and `C`, we do not make use of the notation `x:X` simply to indicate that they are not existing prosumer services and thus they can be either implemented from scratch or partially reused (for the provider part). Then, the second layer contains the BCs to cope with possibly needed middleware-level protocol adaptation, e.g., REST versus SOAP adaptation. It is worth mentioning that SOAP is the default interaction paradigm for the underlying layers. Finally, the last two layers include the Adapter and CD artefacts for adaptation and coordination purposes, respectively. Note that Fig. 2 shows the case in which the participants `B` and `C` are implemented from scratch, and hence BCs together with As are not needed.

The generated artefacts are not always required; rather, it depends on the specified choreography and the characteristics of the existing services (e.g., application-level interaction protocols, interface specifications, middleware-level interaction paradigms) that have been selected to instantiate the roles of the choreography participants. For instance, for this illustrative example, no security policy is specified and, hence, no SF is generated.

3 CHOReVOLUTION Approach

This section describes the CHOReVOLUTION approach for realizing service choreographies by possibly reusing existing services. The approach distinguishes two main phases: “From idea to model” and “From model to runtime”.

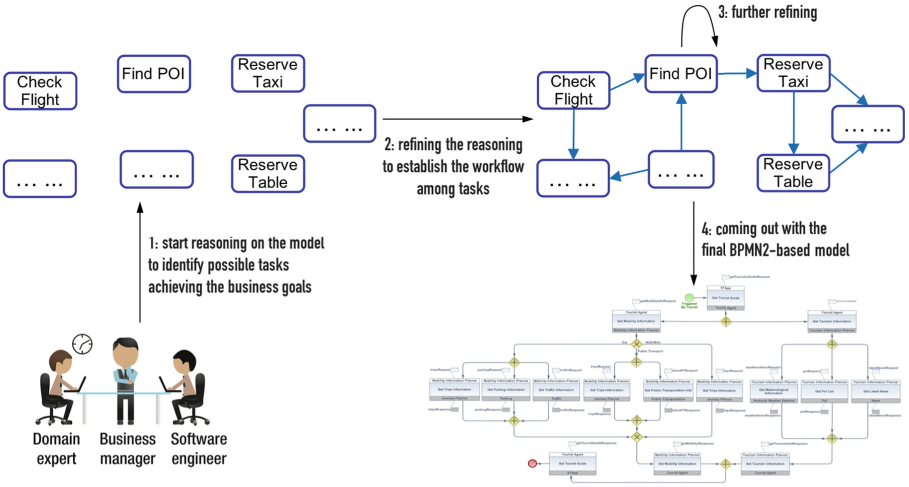

Fig. 3. From idea to model

From Idea to Model. As shown in Fig. 3, system modelers seat together and cooperate to set what are the business goals of the system they have in mind. For instance, a possible goal might be: assisting travelers from arrival, to staying, to departure. For that purpose, system modelers identify the tasks and the participants that will perform them so as to achieve the goal, e.g., reserving a taxi from the local company, purchasing digital tickets at the train station, performing transactions through services based on near field communication in a shop (step 1). Once business tasks have been identified, system modelers specify how the involved participants must collaborate as admissible flows of the identified tasks, hence producing an high-level specification of the system to be (steps 2 and 3). Note that the definition of the high-level specification is not covered by the CHOReVOLUTION approach. Thus, system modelers can use the notation they are more comfortable with. After the complete workflow among tasks has been established, the high-level specification is concretized into a BPMN2 Choreography Diagram (step 4), which, as introduced above, represents the choreography model the IDRE requires to start with in order to realize the specified system.

From Model to Runtime. As shown in Fig. 4, starting from the choreography diagram, the developer interacts with the IDRE in order to generate the code of the needed Binding Components, Adapters and Coordination Delegates, that are used to correctly implement the specified choreography. As already introduced, a service inventory is also accounted for. It contains services published by providers that, for instance, have identified business opportunities in the domain of interest. Providers can be transportation companies, airport retailers, local municipalities, etc., which can be reused in the resulting choreographed system. By exploiting the Enactment Engine provided by the IDRE, the pro-

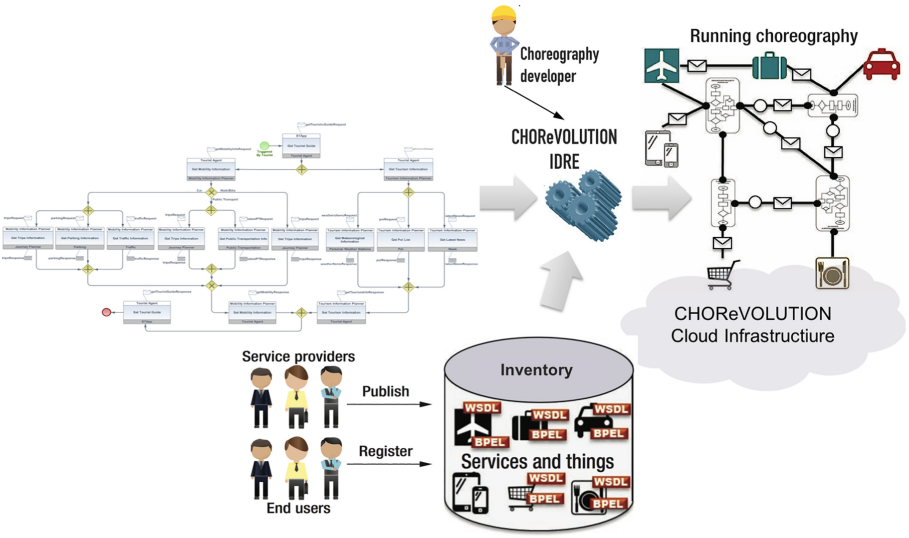

Fig. 4. From model to runtime

duced software artefacts are deployed over the Cloud infrastructure, the resulting choreography is enacted and executed.

4 CHOReVOLUTION Development Process

The CHOReVOLUTION development process consists of a set of core *code generation phases* (see Fig. 5) that takes as input a choreography specification and automatically generates the set of additional software entities previously mentioned. When interposed among the services, these software entities "proxify" the participant services to externally coordinate and adapt their business-level interaction, as well as to bridge the gap of their middleware-level communication paradigms and enforce security constraints.

Validation. This activity validates the correctness of the choreography specification against the constraints imposed by the BPMN2 standard specification. The goal is to check practical constraints concerning both choreography realizability and its enforceability.

Choreography Projection. Taking as input the BPMN2 Choreography Diagram and the related Messages XML schema, this activity automatically extracts all the choreography participants and applies a model-to-model (M2M) transformation to derive the related Participant Models, one for each participant. A participant model is itself a BPMN2 Choreography Diagram. It contains only the choreography flows that involve the considered participant. The generated participant models will be then taken as input by the Coordination Delegate (CD) Generation activity.

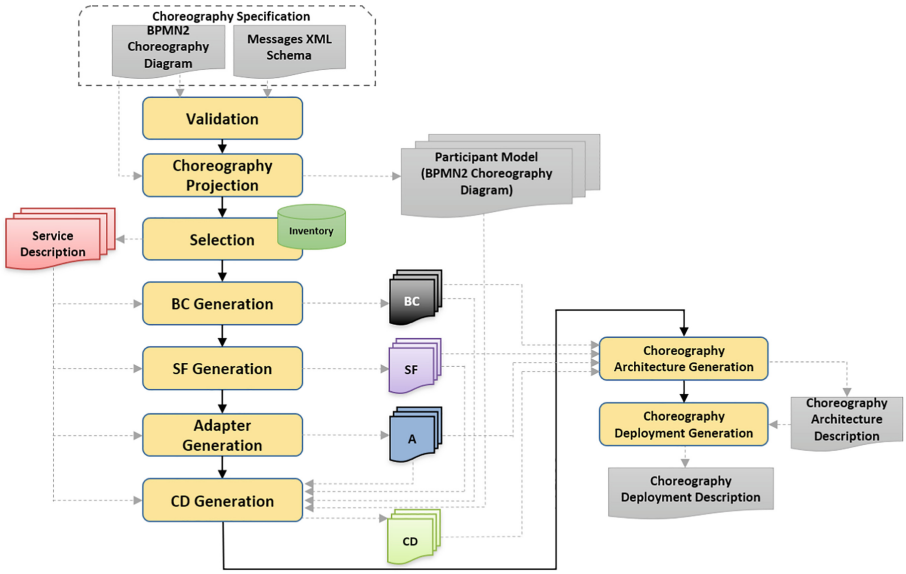

Fig. 5. CHOReVOLUTION development process

Selection. This activity is about querying the Service Inventory in order to select concrete services that can play the roles of the choreography participants. Once the right services have been selected, the related description models will be used to generate the Binding Components (BCs), Adapters (As), and Coordination Delegates (CDs).

BC Generation. BCs are generated when the middleware-level interaction paradigm of a selected service is different from SOAP[5], which is used by the CDs as the middleware-level interaction paradigm.

SF Generation. SFs are generated for those (selected) services having security policies associated. SFs filter the services interactions according to the specified security requirements.

Adapter Generation. When needed, adapters allow to bridge the gap between the interfaces and interaction protocols of the selected services and the ones of the (respective) participant roles they have to play, as obtained via projection. In other words, adapters solve possible interoperability issues due to operation names mismatches and I/O data mapping mismatches (see [6,22]).

CD Generation. CDs are in charge of coordinating the interactions among the selected services so as to fulfill the global collaboration prescribed by the choreography specification, in a fully distributed way (see [2,3,5,7]).

Choreography Architecture Generation. Considering the selected services and the generated BCs, As, and CDs, an architectural description is automatically generated, and a graphic representation of the choreographed system is

[5] http://www.w3.org/TR/soap/.

provided, where all the system's architectural elements and their interdependencies are represented.

Choreography Deployment Generation. The last activity of the development process concerns the generation of the Choreography Deployment Description (called `ChorSpec`) out of the Choreography Architecture model. The deployment description will be used for deploying and enacting the realized choreography.

5 CHOReVOLUTION IDRE

As depicted in Fig. 6, the CHOReVOLUTION IDRE is layered into: a front–end layer (1), a back–end layer (2), and a cloud layer (3).

The Front-end layer (1) consists of two components: a development studio and a web console.

The **CHOReVOLUTION Studio** is an Eclipse-based IDE that allows for (i) designing a BPMN2 Choreography Diagrams; (ii) defining all the details required to instrument the interaction among the services involved in the choreography (e.g., service signatures, identity attributes and roles); (iii) wizarding the code generation phases.

The **CHOReVOLUTION Console** is a web application based on Apache Syncope[6]. It allows to (i) configure, administer, and trigger actions on running services and choreographies; (ii) monitor the execution of a choreography with respect to relevant parameters, such as execution time of choreography tasks, number of messages exchanged, end-to-end deadlines, etc.

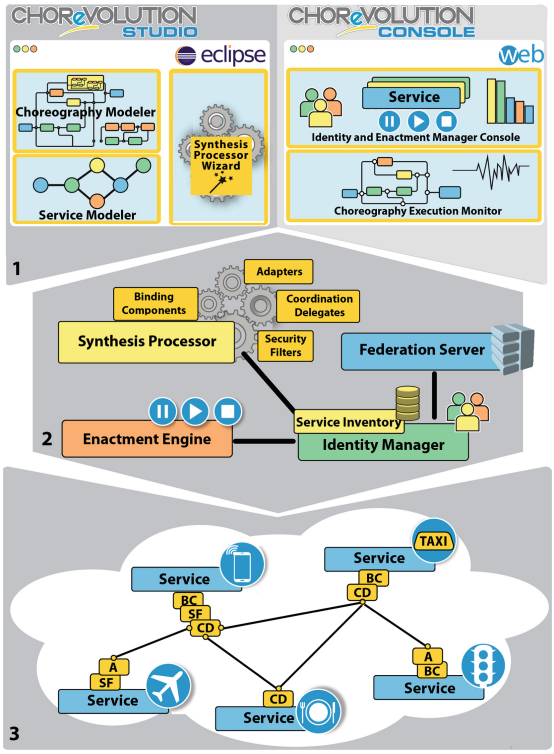

Fig. 6. CHOReVOLUTION IDRE overview

[6] https://syncope.apache.org/.

The Back-end layer (2) consists of the following components.

The **Synthesis Processor** is realized by a set of REST services that implement the model transformations to generate BCs, SFs, CDs, As, the architecture, and the deployment descriptor, as described in previous sections.

The **Enactment Engine (EE)** is a REST API that extends the Apache Brooklyn project[7]. It automatically deploys the choreography according to its deployment description by using the Cloud Layer. The EE also interacts with the Identity Manager to include into the deployment description the actual deployment and runtime details. Then, once a choreography is deployed and running, the EE listens for command requests from the Identity Manager for runtime choreography control. It is worth noticing that, although choreography monitoring and control is performed by centralized IDRE components (e.g., EE and IdM), the realization and running of the choreography is fully distributed into the various artefacts generated by the Synthesis Processor.

The **Federation Server** handles the runtime authentication and authorization for services that uses different security mechanism at the protocol level by storing various credentials on behalf of the caller.

The **Identity Manager (IdM)** is based on Apache Syncope project also. It is responsible for managing users and services. In particular, the IdM is able to query the services for supported application contexts and played roles; force a specific application context for a certain service (put in "maintenance" or disable/enable). The Service Inventory is a sub-component of the IdM. It acts as a central repository for the description models of the services and things that can be used during the synthesis process.

The Cloud layer (3) executes choreography instances on a cloud infrastructure and adapts their execution based on the actual application context.

At execution time, for each choreography, in the CHOReVOLUTION cloud, there are (i) a set of choreography instances at different execution states; (ii) a set of virtual machines executing a custom-tailored mix of services and middleware components to serve different parts of the choreography. Virtual Machines are installed and configured with services according to selectable policies. Due to the fact that EE is based on Apache Brooklyn, the CHOReVOLUTION IDRE can integrate with different Infrastructure as a Service (IaaS) platforms (e.g., Open Stack[8], Amazon EC2[9]).

6 Illustrative Example

This section describes the CHOReVOLUTION IDRE at work on a Smart Mobility and Tourism (SMT) use case. Figure 7 shows the specified BMPN2 choreography diagram. The SMT choreography is used to realize a Collaborative Travel Agent System (CTAS) through the cooperation of several content and service providers, organizations and authorities. It involves a mobile application as an

[7] https://brooklyn.apache.org/.
[8] https://www.openstack.org/.
[9] https://aws.amazon.com.

"Electronic Touristic Guide" that exploits CTAS to provide both smart mobility and touristic information.

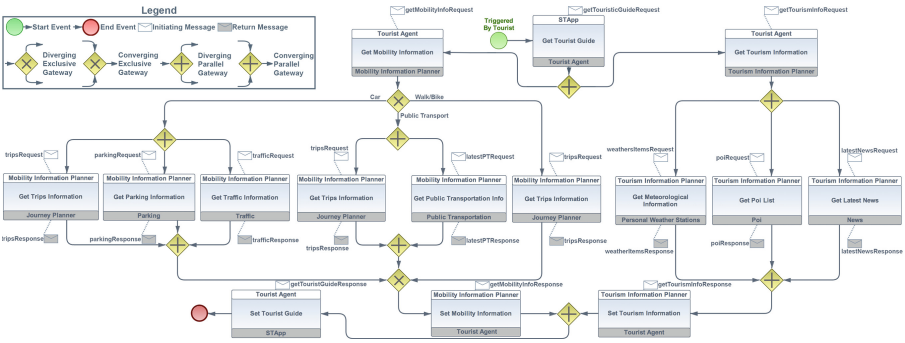

Fig. 7. Smart mobility and tourism choreography

The choreography starts with the mobile application `STApp` detecting the current position of the user, and asking for which type of point of interest to visit and which type of transport mode to use. From this information, `Tourist Agent` initiates two parallel flows in order to retrieve the information required by the "Electronic Touristic Guide" (see the parallel branch with two outgoing arrows after the choreography task Get Tourist Guide). In particular, in the left-most branch of the choreography, `Mobility Information Planner` is in charge of the retrieval of smart mobility information according to the selected transport mode (see the conditional branching), while in the right-most branch, `Tourism Information Planner` is responsible for gathering touristic information. After that, the two parallel flows are joined together to produce the data needed for the "Electronic Touristic Guide" (see the merging branch with two incoming arrows in the bottom side of the choreography). Finally, the guide is shown to the user by means of `STApp`.

In the remainder of this section, the application of the IDRE to the SMT use case is discussed by distinguishing the actions performed by the two possible types of users: service providers and choreography developers. A user guide to replicate the example can be found at https://github.com/chorevolution/CHOReVOLUTION-IDRE/wiki/User-Guide.

Service Provider. A service provider uses the IDRE to publish the description models of the services into the Service Inventory. The IDRE allows to deal with heterogeneous services. It provides a uniform description for any service, given by means of the Generic Interface Description Language (GIDL) [12] or the WSDL[10] in case of SOAP services. GIDL supports interface description for any kind of possible services (e.g., REST services). As introduced above, the published

[10] https://www.w3.org/TR/wsdl20-primer/.

services are selected in order to play the participants roles of a choreography. Then, the next phases will use the services' models to generate BCs, SFs, CDs, and As.

Referring to the SMT example, the service provider has to create a Service/Thing project inside the CHOReVOLUTION Studio by using a GIDL description for the following services: `Journey Planner`, `Parking`, `Traffic`, `Public Transportation`, `Personal Weather Stations`, `Poi` and `News`.

Choreography Developer. A developer uses the CHOReVOLUTION Studio to model a choreography and to realize it. The developer has to create a CHOReVOLUTION Synthesis project. Then, she models the BPMN2 choreography diagram by using the Eclipse BPMN2 choreography modeler[11] embedded in the Studio. Afterwards, the developer starts the synthesis process. The first two activities of the process (i.e., Validation and Choreography Projection) do not require any user interaction. The other activities are supported by suitable wizards, as discussed in the following.

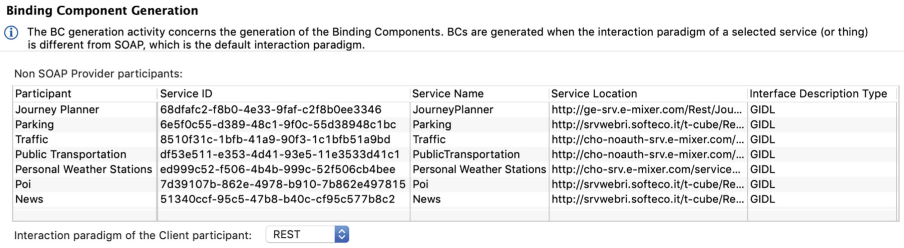

Fig. 8. BC generation activity

- *Selection.* For each participant, the developer selects the corresponding concrete service, as published into the Service Inventory. For instance, for the SMT choreography, the above seven mentioned services.
- *BC Generation.* Figure 8 shows the wizard that is used to configure the BCs generator for those selected services that do not rely on SOAP. Considering the SMT example, all the selected services are REST services. Thus, in Fig. 8, they are all listed in the wizard together with their GIDL description.
- *SF Generation.* None of the services for the SMT choreography defines security policies. Therefore, the SF Generation step is skipped.
- *Adapter Generation.* We recall that some mismatches can arise due to possible heterogeneities between the interfaces of the abstract services in the specification and the ones of the concrete services selected from the inventory (e.g., operation names mismatches and I/O data mapping mismatches).

[11] https://www.eclipse.org/bpmn2-modeler/.

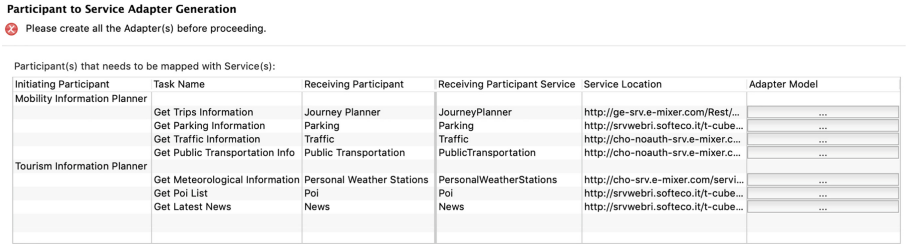

Fig. 9. Adapter generation activity

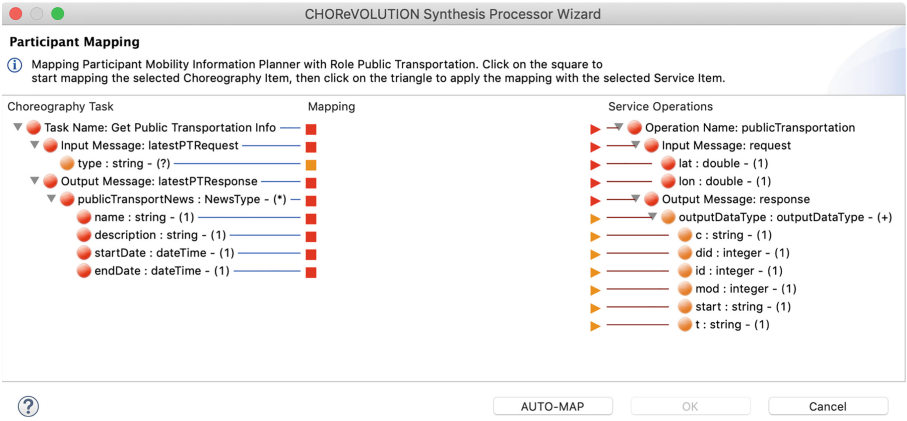

Fig. 10. Adapter mapping

Regarding the SMT choreography, all the selected services exhibit some mismatches with respect to the corresponding choreography participants. The Adapter Generation wizard asks the developer for specifing the needed adaptation logic. In particular, the wizard shows all the choreography tasks that require adaptation, they are grouped by their initiating participant, see the leftmost column in Fig. 9. By clicking on the button labeled with "..." a new dialog window is opened, as shown in Fig. 10.

At this stage, the developer can map task messages to service operations messages. The elements identified with the red shapes are mandatory to be mapped, whereas those in orange are optional. In order to ease the mapping definition, the wizards provides a "`AUTO-MAP`" functionality to automatically generate the mappings by performing a syntactic binding to be refined afterwards.

–*CD Generation.* The last step of the wizard concerns the Coordination Delegates generation (Fig. 11).

If needed, the developer can specify "causality" correlations between different choreography tasks. Two tasks are correlated when they respectively represent an asynchronous request (the first task) and the subsequent callback (the second

task). This also means that the initiating (resp., receiving) participant of the first task must be the receiving (resp., initiating) one of the second task. Considering the SMT choreography, the mobile application starts the choreography by sending the user preferences (current position, type of transport mode to use, etc.) and finally it gets back all the information needed to show an "Electronic Touristic Guide" to the user. Thus, the developer has to specify a correlation between the task `Get Tourist Guide` and the task `Set Tourist Guide`.

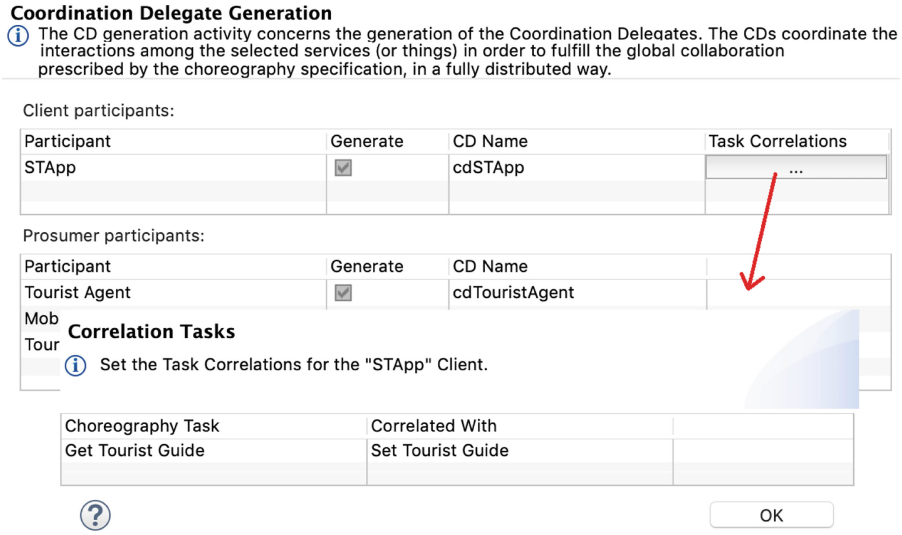

Fig. 11. CD generation activity

By clicking on the Finish button, all the software artefacts (BCs, SFs, ADs, CDs) are generated. In addition, for each participant that acts as both an initiating participant in some task and a receiving participant in a different task (i.e., Tourist Agent, Mobility Information Planner, and Tourism Information Planner), the skeleton code of its business logic is generated to be then completed by the developer. This is the construction logic for the messages sent by the participant.

Figure 12 shows the code completed by the developer for building the message `getMobilityInfoResponse` (see local variable `result`). The implemented logic starts with the retrieval of the message `tripsResponse` sent by Journey Planner within the task `Get Trips Information` (line 297). The content of this message is used to set the trip information of `getMobilityInfoResponse` (line 298). Then `getMobilityInfoRequest` sent by `Tourist Agent` is retrieved (lines 300–301). Based on the transportation mean chosen by the user, which is contained in the `transportMode` element of the message, different data can be used to construct the response message `getMobilityInfoResponse`.

MobilityInformationPlannerServiceImpl.java

```
@Override
public GetMobilityInfoResponse createGetMobilityInfoResponse(
        ChoreographyInstanceMessages choreographyInstanceMessages, String choreographyTaskName,
        String receiverParticipantName) {
    GetMobilityInfoResponse result = new GetMobilityInfoResponse();
    TripsResponse tripsResponse = (TripsResponse) choreographyInstanceMessages
            .getMessageSentFromParticipant("tripsResponse", "Journey Planner", "Get Trips Information");
    result.setTrip(tripsResponse);
    GetMobilityInfoRequest getMobilityInfoRequest = (GetMobilityInfoRequest) choreographyInstanceMessages
            .getMessageSentFromParticipant("GetMobilityInfoRequest", "Tourist Agent","Get Mobility Information");
    if(getMobilityInfoRequest.getTransportMode().equals(Modes.CAR)) {
        ParkingResponse parkingResponse = (ParkingResponse) choreographyInstanceMessages
                .getMessageSentFromParticipant("parkingResponse", "Parking", "Get Parking Information");
        result.getParkings().addAll(parkingResponse.getParkings());
        TrafficResponse trafficResponse = (TrafficResponse) choreographyInstanceMessages
                .getMessageSentFromParticipant("trafficResponse", "Traffic", "Get Traffic Information");
        result.getTrafficInfos().addAll(trafficResponse.getTrafficInfos());
    }
    if(getMobilityInfoRequest.getTransportMode().equals(Modes.PUBLIC_TRANSPORT)) {
        LatestPTResponse latestResponse = (LatestPTResponse) choreographyInstanceMessages
                .getMessageSentFromParticipant("latestPTResponse", "Public Transportation",
                        "Get Public Transportation Info");
        result.getPublicTransportInfo().addAll(latestResponse.getPublicTransportNews());
    }
    return result;
}
```

Fig. 12. Prosumer business logic implementation

Choreography Architecture Generation - Finally, considering the selected services and the generated BCs, SFs, ADs, and CDs, an architectural description is automatically generated in both a textual and a graphical form.

7 Conclusion

This paper has presented the CHOReVOLUTION IDRE, an integrated platform for developing, deploying, executing and monitoring choreography-based distributed applications.

In this tutorial paper, an industrial use case, in the Smart Mobility and Tourism domain, has been used to show the CHOReVOLUTION IDRE at work. The industrial partners that provided us with the use case have experienced with its modeling and automatic development and enactment, by using the IDRE. While interacting with the IDRE software development facilities and wizards discussed in this paper, the involved industrial partners experienced a significant time decrease with respect to realizing the use case by exploiting their daily development approaches. Their feedbacks on that indicate that the CHOReVOLUTION IDRE has a great potential in developing choreography-based applications and the use case got a full benefit from it.

More pilots and development cases will allow to consolidate the technical maturity of the product and pose the basis for a commercial validation.

Acknowledgments. Supported by: (i) EU H2020 Programme grant no. 644178 (CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), (ii) the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (INCIPICT), and (iii) the SISMA national PRIN project (contract no. 201752ENYB).

References

1. Autili, M., Inverardi, P., Tivoli, M.: Automated synthesis of service choreographies. IEEE Softw. **32**(1), 50–57 (2015). https://doi.org/10.1109/MS.2014.131
2. Autili, M., Inverardi, P., Perucci, A., Tivoli, M.: Synthesis of distributed and adaptable coordinators to enable choreography evolution. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III. Assurances. LNCS, vol. 9640, pp. 282–306. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-74183-3_10
3. Autili, M., Inverardi, P., Tivoli, M.: Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates. Sci. Comput. Program. **160**, 3–29 (2018). https://doi.org/10.1016/j.scico.2017.10.010
4. Autili, M., Di Ruscio, D., Di Salle, A., Inverardi, P., Tivoli, M.: A model-based synthesis process for choreography realizability enforcement. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 37–52. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_4
5. Autili, M., Ruscio, D.D., Salle, A.D., Perucci, A.: Choreosynt: enforcing choreography realizability in the future internet. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, 16–22 November 2014, pp. 723–726 (2014). https://doi.org/10.1145/2635868.2661667
6. Autili, M., Salle, A.D., Gallo, F., Pompilio, C., Tivoli, M.: Model-driven adaptation of service choreographies. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, pp. 1441–1450 (2018). https://doi.org/10.1145/3167132.3167287
7. Autili, M., Salle, A.D., Gallo, F., Pompilio, C., Tivoli, M.: On the model-driven synthesis of evolvable service choreographies. In: 12th European Conference on Software Architecture: Companion Proceedings, ECSA, pp. 20:1–20:6 (2018). https://doi.org/10.1145/3241403.3241425
8. Basu, S., Bultan, T.: Choreography conformance via synchronizability. In: Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28–April 1 2011, pp. 795–804 (2011). https://doi.org/10.1145/1963405.1963516
9. Basu, S., Bultan, T.: Automatic verification of interactions in asynchronous systems with unbounded buffers. In: ACM/IEEE International Conference on Automated Software Engineering, ASE 2014, Vasteras, Sweden - 15–19 September 2014, pp. 743–754 (2014). https://doi.org/10.1145/2642937.2643016
10. Basu, S., Bultan, T.: Automated choreography repair. In: Stevens, P., Wąsowski, A. (eds.) FASE 2016. LNCS, vol. 9633, pp. 13–30. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49665-7_2
11. Basu, S., Bultan, T., Ouederni, M.: Deciding choreography realizability. In: Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, 22–28 January, pp. 191–202 (2012). https://doi.org/10.1145/2103656.2103680
12. Bouloukakis, G.: Enabling emergent mobile systems in the IoT: from middleware-layer communication interoperability to associated QoS analysis. Ph.D. thesis, Inria, Paris, France (2017)
13. Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M., Patrizi, F.: Automatic service composition and synthesis: the Roman model. IEEE Data Eng. Bull. **31**(3), 18–22 (2008)

14. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Proceedings of 40th Symposium on Principles of Programming Languages, pp. 263–274 (2013). https://doi.org/10.1145/2429069.2429101
15. European Commission: Digital agenda for Europe - Future Internet Research and Experimentation (FIRE) initiative (2017). https://ec.europa.eu/digital-single-market/en/future-internet-research-and-experimentation
16. Gössler, G., Salaün, G.: Realizability of choreographies for services interacting asynchronously. In: Arbab, F., Ölveczky, P.C. (eds.) FACS 2011. LNCS, vol. 7253, pp. 151–167. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35743-5_10
17. Güdemann, M., Poizat, P., Salaün, G., Ye, L.: Verchor: a framework for the design and verification of choreographies. IEEE Trans. Serv. Comput. **9**(4), 647–660 (2016). https://doi.org/10.1109/TSC.2015.2413401
18. Lanese, I., Montesi, F., Zavattaro, G.: The evolution of Jolie: from orchestrations to adaptable choreographies. In: De Nicola, R., Hennicker, R. (eds.) Software, Services, and Systems. LNCS, vol. 8950, pp. 506–521. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15545-6_29
19. Poizat, P., Salaün, G.: Checking the realizability of BPMN 2.0 choreographies. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, 26–30 March 2012, pp. 1927–1934 (2012). https://doi.org/10.1145/2245276.2232095
20. Salaün, G.: Generation of service wrapper protocols from choreography specifications. In: Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10–14 November 2008, pp. 313–322 (2008). https://doi.org/10.1109/SEFM.2008.42
21. Salaün, G., Bultan, T., Roohi, N.: Realizability of choreographies using process algebra encodings. IEEE Trans. Serv. Comput. **5**(3), 290–304 (2012). https://doi.org/10.1007/978-3-642-00255-7_12
22. Di Salle, A., Gallo, F., Perucci, A.: Towards adapting choreography-based service compositions through enterprise integration patterns. In: Bianculli, D., Calinescu, R., Rumpe, B. (eds.) SEFM 2015. LNCS, vol. 9509, pp. 240–252. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-49224-6_20