Model-driven adaptation of service choreographies

Marco Autili
University of L'Aquila
L'Aquila, Italy
marco.autili@univaq.it

Amleto Di Salle
University of L'Aquila
L'Aquila, Italy
amleto.disalle@univaq.it

Francesco Gallo
University of L'Aquila
L'Aquila, Italy
francesco.gallo@univaq.it

Claudio Pompilio
University of L'Aquila
L'Aquila, Italy
claudio.pompilio@graduate.univaq.it

Massimo Tivoli
University of L'Aquila
L'Aquila, Italy
massimo.tivoli@univaq.it

ABSTRACT

Service choreographies represent a powerful and flexible approach to compose software services in a fully distributed way. A key enabler for the actual realization of choreographies is the ability to automatically compose services, and perform exogenous coordination and adaptation of their interaction. This is a nontrivial and error prone task. Automatic support for realizing choreographies is needed. In this paper we focus on adapter generation and describe our novel approach to the synthesis of service Adapters. When needed, adapters permit to correctly bind concrete services to (abstract) choreography roles by solving possible protocol mismatches. Enterprise Integration Patterns are used as adaptation primitives and composed to realize complex adaptation policies.

CCS CONCEPTS

• **Applied computing → Service-oriented architectures**; **Enterprise application integration**;

KEYWORDS

Service-Oriented Computing; Service Choreography; Model-driven; Adaptation; Enterprise Integration Pattern.

ACM Reference Format:
M. Autili, A. Di Salle, F. Gallo, C. Pompilio and M. Tivoli. 2018. . In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* , Article 4, 10 pages. https://doi.org/10.1145/3167132.3167287

1 INTRODUCTION

Service choreographies represent a powerful and flexible approach to compose software services in a fully distributed way. Service choreographies have been around since many years, and many valuable (mostly theoretic) approaches have been proposed [9, 15, 17, 20, 27] (just to mention a few).

With the objective of bringing the adoption of choreographies to the development practices currently adopted by IT companies, our research and development activity has been focused on practical and automatic approaches to support the realization of model-based reuse-oriented service choreographies [1–7]. During the last decade, this research and development activity has been funded (in particular) by two EU projects: the FP7 CHOReOS and its follow-up H2020 CHOReVOLUTION[1].

The need for service choreographies was recognized in the Business Process Modeling Notation version 2.0[2] (BPMN2), which introduced *Choreography Diagrams* to offer choreography modeling constructs. A choreography diagram models peer-to-peer communication by defining a multi-party message-passing protocol that, when put in place by the concrete services that realize the roles of the choreography, permits to reach the overall choreography objectives in a fully distributed way. In BPMN2 choreography diagrams, a participant role models the expected behaviour (e.g., the expected interaction protocol) that a concrete service should support in order to be able to play the role of the participant in the choreography. When third-party participants are involved, usually black-box services to be reused, one of the main problems to be solved when realizing choreographies is *automatic realizability enforcement*. It can be informally phrased as follows: given a choreography specification and a set of existing services to be reused, externally coordinate their interaction so to fulfill the collaboration prescribed by the choreography specification. In order to address this problem from a practical point of view, both coordination and adaptation issues must be solved automatically (and indeed, also security and middleware-level communication issues[3]). Thus, a key enabler for the actual realization of choreographies is the ability to automatically compose services, and perform exogenous coordination and adaptation of their interaction. However, in a distributed setting, obtaining the coordination and adaptation logic required to realize a choreography is nontrivial and error prone. As a matter of fact, automatic support for realizing choreographies is needed.

Towards addressing these challenges, in this paper we briefly describe our overall approach to the automatic synthesis of service choreographies. Then, we focus on the novel contribution and describe in detail the approach we developed to automatically synthesize service Adapters. When needed, adapters permit to correctly bind concrete services to (abstract) choreography roles.

[1]www.choreos.eu – www.chorevolution.eu
[2]http://www.omg.org/spec/BPMN/2.0.2/
[3]The treatment of these issues is out of scope for this paper. They have been mentioned here only for completeness with respect to the overall approach (described in Section 2) we are developing within the CHOReVOLUTION project.

Specifically, adapters solve possible protocol mismatches between services and choreography roles by employing Enterprise Integration Patterns [18] (EIP) as adaptation primitives and composing them to realize the required adaptation logic.

The paper is structured as follow. Section 2 provides a brief overview of the overall synthesis process we have implemented in the CHOReVOLUTION project. Section 3 introduces an explanatory example. Section 4 focus on the novel approach to choreography adaptation, and Section 5 describes it at work on the explanatory example. Related work is discussed in Section 6, and conclusions are given in Section 7.

2 SETTING THE CONTEXT

The synthesis process consists of a set of core *code generation phases* (see Figure 1) that take as input a choreography specification together with a set of concrete services as possible candidates to play the choreography roles and automatically generates a set of additional software entities. When interposed among the services according to a predefined architectural style (see Figure 2), 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. It is worth to clarify that the additional software entities are generated and interposed among the services only if strictly needed, according to possible coordination and adaptation issues that have been identified, as well as middleware level binding issues and security requirements.

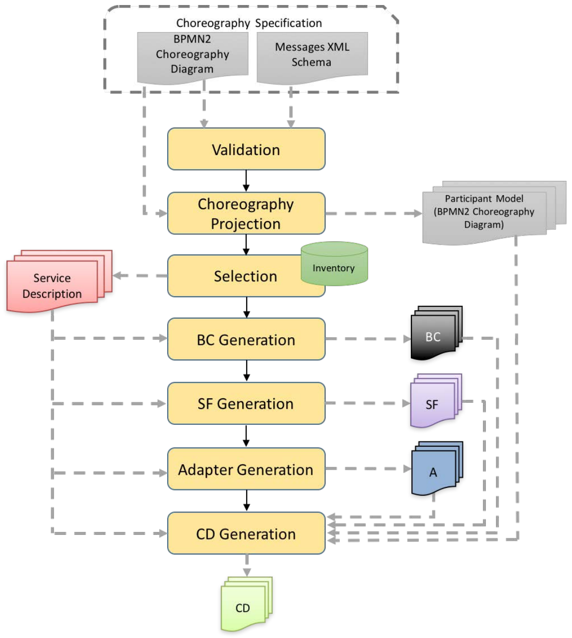
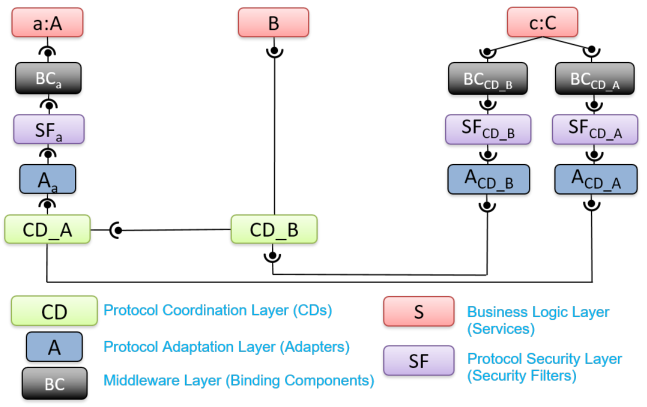


Figure 1: Synthesis process

For what concerns coordination and adaptation, coordination entities, called Coordination Delegates (CDs), enforce the collaboration prescribed by the choreography specification through distributed protocol coordination. Importantly, the CD coordination logic is concrete service independent, meaning that it is synthesized by considering the expected interaction protocol specified by the choreography diagram for the (abstract) participant roles instead of the one of the concrete services. This allows our approach to realize separation of concerns, hence possibly reusing the synthesized coordination logic when (late) binding different concrete services to the choreography roles. Thus, when needed, service Adapters are synthesized in order to correctly bind concrete services to abstract roles. As already introduced, adapters solve possible protocol mismatches between services and choreography roles by employing Enterprise Integration Patterns [18] (EIP) as adaptation primitives, and composing them to realize the required adaptation logic.

Concerning middleware-level binding issues and enforcing security requirements, additional components are generated, namely, Binding Components (BCs) and Security Filters (SFs),


a:A
B
c:C
BC_a
BC_CD_B
BC_CD_A
SF_a
SF_CD_B
SF_CD_A
A_a
A_CD_B
A_CD_A
CD_A
CD_B
CD
Protocol Coordination Layer (CDs)
A
Protocol Adaptation Layer (Adapters)
BC
Middleware Layer (Binding Components)
S
Business Logic Layer (Services)
SF
Protocol Security Layer (Security Filters)
Figure 2: Architectural Style (a sample instance of)

In order to better collocate the adapter generation approach within the overall picture, we now give a step-by-step description of the overall synthesis process.

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 (see [4, 8] and related work in [9, 10, 15, 17, 20, 27]).

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.

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 BCs, SFs, Adapters, and CDs, as per the following steps.

BC Generation – BCs are generated when the middleware-level interaction paradigm of a selected service is different from SOAP[4], which is used by the coordination delegates as the middleware-level interaction paradigm.

[4]www.w3.org/TR/soap/

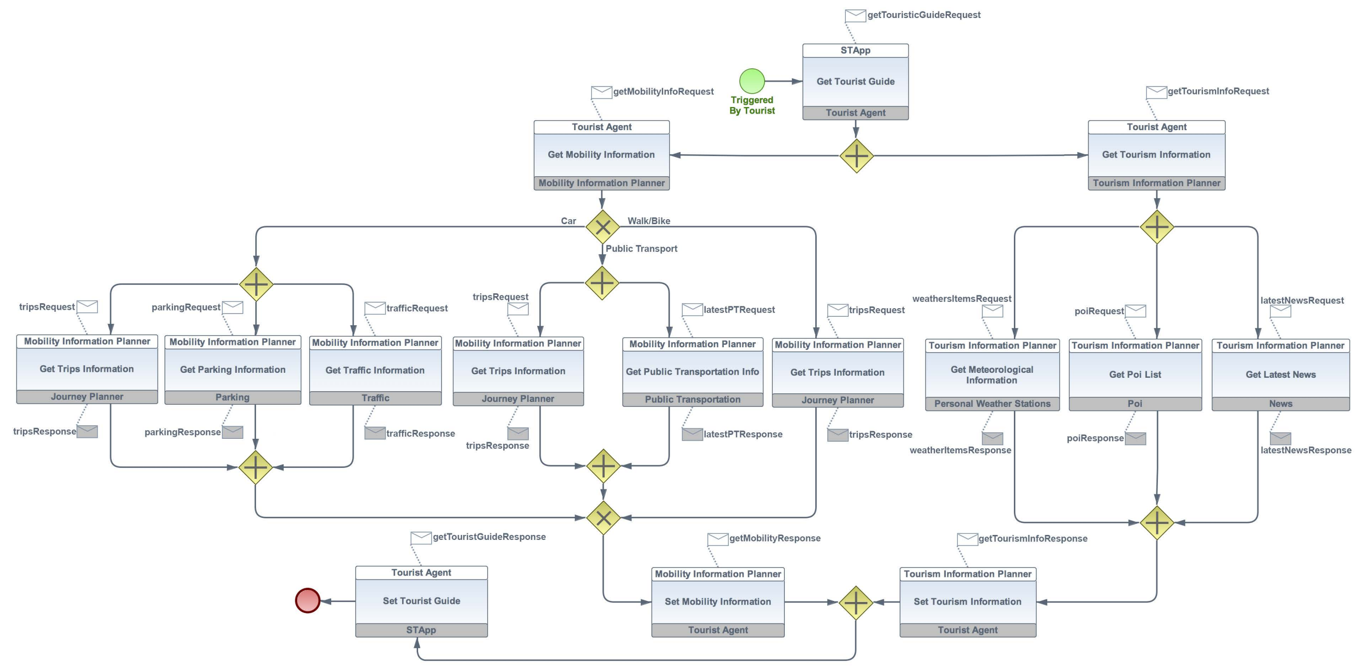


Figure 3: Smart and Mobility Tourism Choreography Specification

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. Adapters are automatically synthesized to mediate the end-to-end interaction Service-to-CD and CD-to-Service, also taking into account possible SFs and BCs in between (see Figure 2). In other words, adapters solve possible interoperability issues due to operation names mismatches and I/O data mapping mismatches.
CD Generation – CDs are in charge of coordinating the interactions among the selected services so as to allow only those (possibly parallel) sequences of message exchanges that are prescribed by the choreography specification, in a fully distributed way.

As already anticipated, all the generated artefacts are finally deployed and connected according to a predefined layered architectural style, a sample instance of which is shown in Figure 2.

While performing all the process phases in Figure 1 (including the generation of the final architecture and the final deployment step – not shown in the figure), developers are supported by a wizard that guide them while selecting the required inputs and producing the corresponding output artefacts. The wizard is offered as part of a customization of the Eclipse platform that we have implemented for realizing the overall approach[5].

From now on, we focus on the novel contribution of this paper, which concerns the adapter generation phase.

[5]www.chorevolution.eu/bin/view/Documentation/Download

3 CASE STUDY

This section introduces a case study from the CHOReVOLUTION project named Smart and Mobility Tourism (SMT). It will be used as a running example in the remainder of the paper.

Figure 3 shows the SMT choreography diagram, specified through the BPMN2 Choreography Diagram. As already introduced, a choreography diagram specifies the way the choreography participants exchange information (messages) from a global point of view. The main element of a choreography diagram is the choreography task (e.g., `Get Tourist Guide` task on top of Figure 3). Graphically, BPMN2 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. The white box denotes the initiating participant that decides when the interaction takes place. A task is an atomic activity that represents an interaction by means of one or two (request and optionally response) message exchanges (`getTouristGuideRequest`) between two participants (`STApp` and `Tourist Agent`).

The main scope of the SMT scenario is to realize a Collaborative Travel Agent Systems (CTAS) through the cooperation of several content and service providers, organizations and authorities. The SMT use case envisages a mobile application as an “Electronic Touristic Guide”, by exploiting the CTAS in order to provide both smart mobility and touristic information. The scenario starts with the mobile application `STApp` detecting the current position of the user, and asking which type of point of interest to visit and which type of transport mode to use. From this information, the choreography initiates two main parallel execution flows in order to retrieve the information required by the “Electronic Touristic Guide” (see the parallel branch represented as a rhombus marked with a “+” with two outgoing arrows after the choreography task `Get Tourist`

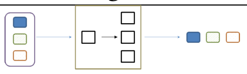
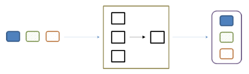
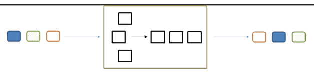
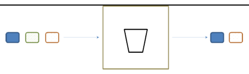
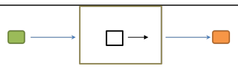
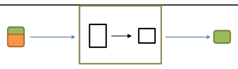
Message Routing EIP		
Name	**Description**	**Figure**
Splitter	It splits a message into several ones, and sends the resulting messages to be processed independently	
Aggregator	It receives multiple messages and combines them into a single message. It is stateful and it must buffer the messages to be aggregated and determine when they are completed	
Resequencer	It collects and re-orders messages and put them into an output channel in the specified order. Similarly to the Aggregator, it is stateful	
Message Filter	It decides whether a message should be passed along or dropped based on some criteria respect to the header and/or content of the message	
Message Transformation EIP		
Message Translator	It converts a message from a format to another one	
Content Filter	It permits to remove some elements from the message content	

Table 1: Considered EIP

`Guide`). In particular, the left-most branch of the choreography is in charge of the retrieval of smart mobility information according to the selected transport mode (see the conditional branching represented as a rhombus marked with a "×"), while the right-most branch is responsible to gather touristic information. Finally, the two main parallel flows are joined together on the production of the data needed for the "Electronic Touristic Guide" (see the merging branch represented as a rhombus marked with a "+" with two incoming arrows in the lower part of the choreography), that is then shown to the user by means of `STApp`.

4 APPROACH

In this section we describe the proposed model-driven approach, by presenting the underlying metamodel as first. Then, the transformation rules to generate the adapters are described.

Adapters are needed when the interaction protocol of the selected concrete services are different from the abstract descriptions of the participants they play in the BPMN2 choreography specification. Thus, the operations signature plus interaction via messages exchange, may need to be adapted to the (abstract) interface of the choreography participants. This requires to implement a suitable notion of matching between protocols by means of (possibly complex) data mappings over both operation names and I/O messages, and their flows. The Adapter metamodel is used to represent this matching. It considers the subset of Enterprise Integration Patterns (EIPs) [18] that belong to the class of Message Routing Patterns and patterns from the class Message Transformation, in Table 1. The former class of patterns is used to decouple a message source from the ultimate destination of the message following specific message routing policies while the latter is used to deal with message data format mismatches.
The considered EIPs permit two types of adaptation.

Flow-driven adaptation – realized by means of Message Routing EIPs, it is used to adapts sequences of messages by operating at the granularity of messages, without touching the content of the messages

Data-driven adaptation – realized by means of Message Transformation EIPs, it is used to adapt the content of a single message exchange. This type of adaptation operates at the granularity of the data items constituting the message and their types.

Complex flow-driven and data-driven adaptation policies can be achieved as chains of both Message Transformation and Message Routing EIPs.

4.1 Adapter Metamodel

The adapter metamodel in Figure 4 specifies data mappings between the messages specified for the choreography tasks and the request/response messages sent/received through concrete service operations.

As anticipated in Section 2, adapters are responsible for mediating two kinds of interactions: CD-to-Service and Service-to-CD. Due to this reason the adapter metamodel has an attribute `type` inside the `AdapterModel` metaclass. Its possible values can be either `choreography_to_service` in case of a CD-to-Service mediation, or `service_to_choreography` in case of a Service-to-CD mediation. The core of the adapter metamodel is built around four main concepts, namely *operations*, *messages*, *data type*, and *enumeration*, which reflect the relations needed to express the aforementioned EIPs.

Operation and Message Relations – permit to specify Message Routing patterns by mapping one or more choreography tasks to one or more service operations, together with their messages. The Splitter EIP is realized by a one-to-many operation and message mapping, i.e., one choreography task is mapped to more than one concrete service operation. The Aggregator EIP is realized by a many-to-one mapping. Consequently, the Resequencer EIP is realized by a many-to-many mapping. The desired order of the permutation realized by the resequencer can be specified by using

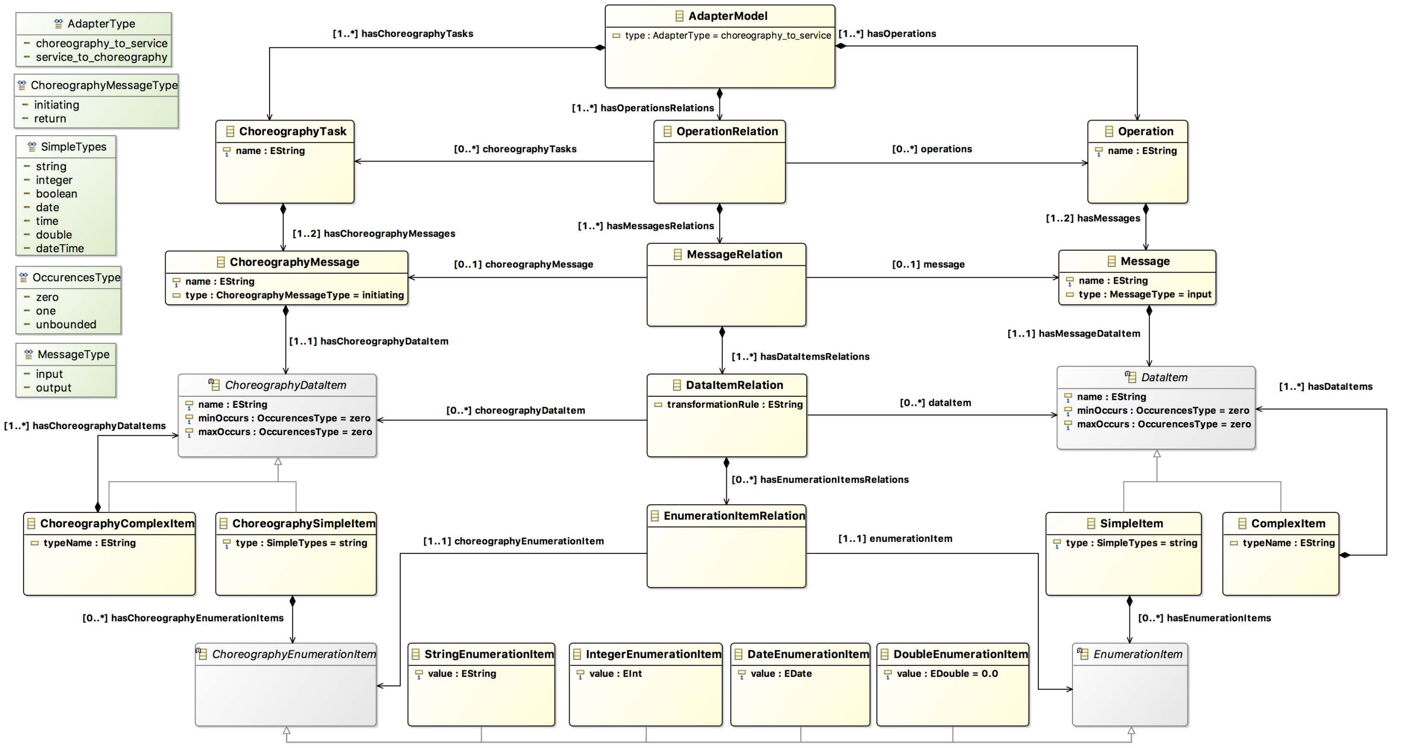


Figure 4: Service-Role Adapter metamodel

a property of the message relation (not shown in the figure). The Message Filter EIP can be realized by a many-to-many operation mapping, combined with a many-to-zero message mapping. The latter relation serves to drop the messages that must be filtered out.

Message Data Type and Enumeration Item Relations – permit to specify Message Transformation patterns by associating choreography message data items with the related service message data items. The Message Translator EIP can be realized by means of plain correspondences among the involved data items or transformation rules that convert choreography message data items into service message data items. Enumeration item relations are specifically used to map the entries of an enumerated type to the entries of another enumerated type. Finally, the absence of a `DataItemRelation` on a specific message item denotes a Content Filter that drops the message item.

4.2 Rule–based adapter generation

The adapter generation phase is carried on by applying a number of rules, capable of realizing both Flow-driven adaptation and Data-driven one. In particular, the former is realized through **Rule 1 to Rule 7**; the latter is realized by means of **Rule 8** and **Rule 9**. These rules are applied by navigating an adapter model in a top-down fashion. This means that the rules involving the Operation and Message Relations are applied first, resulting in the Flow-driven adaptation. The rules involving Message Data Type and Enumeration Item Relations are applied only after, resulting in the Data-driven adaptation.

In the following, all the rules are described by considering their possible situations as reported in Figure 5.

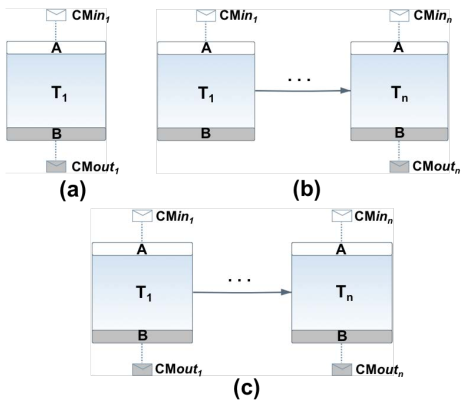


Figure 5: Choreographies reference scenarios

The situation in Figure 5.(a) underlies the definition of the rules 1 to 3. Suppose that, during the selection phase, a concrete service has been selected to play the role of the participant B. Let us also suppose that this concrete service has $n > 1$ operations, $op\_1, \ldots, op\_n$, that match the choreography task $T1$. Referring to Figure 6.(a), in the resulting adapter model, the `OperationRelation` associates the task $T1$ to the service operations $op\_1, \ldots, op\_n$. Note that the value of the attribute type is set to `choreography_to_service`. Exploiting this adapter model, rules 1 to 3 can be described as follows.

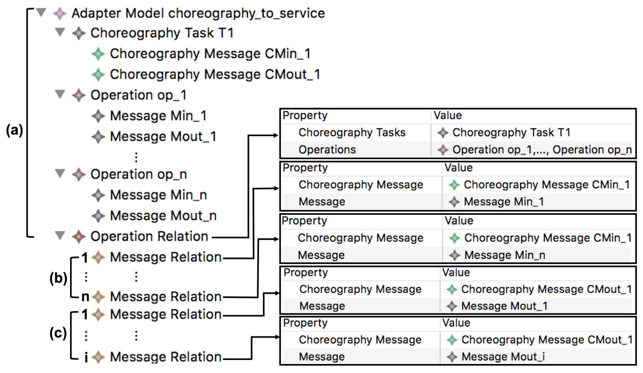

Adapter Model choreography_to_service
Choreography Task T1
Choreography Message CMin_1
Choreography Message CMout_1
Operation op_1
Message Min_1
Message Mout_1
Operation op_n
Message Min_n
Message Mout_n
Operation Relation
1 Message Relation
n Message Relation
1 Message Relation
i Message Relation
(a)
(b)
(c)
Property | Value
Choreography Tasks | Choreography Task T1
Operations | Operation op_1,..., Operation op_n
Choreography Message | Choreography Message CMin_1
Message | Message Min_1
Choreography Message | Choreography Message CMin_1
Message | Message Min_n
Choreography Message | Choreography Message CMout_1
Message | Message Mout_1
Choreography Message | Choreography Message CMout_1
Message | Message Mout_i

Figure 6: Adapter model – rules 1 and 2

[Rule 1] Splitter choreography_to_service – A Splitter is generated if the `OperationRelation` has n `MessageRelations` involving the initiating message $CMin\_1$ of $T1$, and the input messages $Min\_1, \ldots, Min\_n$ of the `Operations` $op\_1, \ldots, op\_n$, respectively (see Figure 6.(b)). The generated Splitter receives the choreography message $CMin\_1$ as input and splits it into the messages $Min\_1, \ldots, Min\_n$.

[Rule 2] Aggregator choreography_to_service – An Aggregator is generated if the `OperationRelation` has $1 < i \leq n$ `MessageRelations` that involve the return message $CMout\_1$ of $T1$, and the output messages $Mout\_1, \ldots, Mout\_i$ of the `Operations` $op\_1, \ldots, op\_i$ (see Figure 6.(c)). The generated Aggregator receives the messages $Mout\_1, \ldots, Mout\_i$ as input and aggregates them into $CMout\_1$.

Still considering the choreography task in Figure 5.(a), let us suppose that, during the selection phase, a concrete service has been selected to play the role of the participant A. The concrete service has $n > 1$ operations, $op\_1, \ldots, op\_n$, that match the choreography task $T1$. Figure 7 shows the related adapter model where the `OperationRelation` associates the task $T1$ to the operations $op\_1, \ldots, op\_n$. Note that now the value of the attribute type is set to `service_to_choreography`. Rule 3 follows.

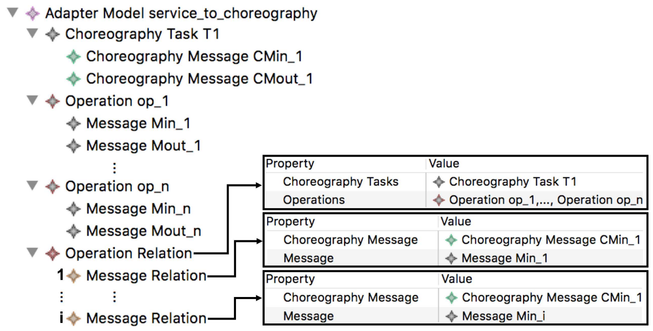

Adapter Model service_to_choreography
Choreography Task T1
Choreography Message CMin_1
Choreography Message CMout_1
Operation op_1
Message Min_1
Message Mout_1
Operation op_n
Message Min_n
Message Mout_n
Operation Relation
1 Message Relation
i Message Relation
Property | Value
Choreography Tasks | Choreography Task T1
Operations | Operation op_1,..., Operation op_n
Choreography Message | Choreography Message CMin_1
Message | Message Min_1
Choreography Message | Choreography Message CMin_1
Message | Message Min_i

Figure 7: Adapter model – rule 3

[Rule 3] Aggregator service_to_choreography – An Aggregator is generated if the `OperationRelation` has $1 < i \leq n$ `MessageRelations` that involve the initiating message $CMin\_1$ of $T1$, and the input messages $Min\_1, \ldots, Min\_i$ of the `Operations` $op\_1, \ldots, op\_i$. The generated Aggregator receives the messages $Min\_1, \ldots, Min\_i$ as input and aggregates them into $CMin\_1$.

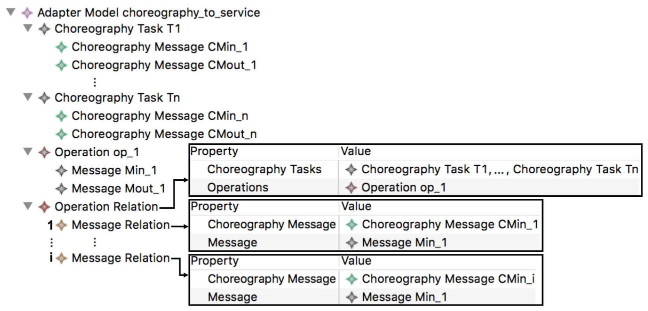

Adapter Model choreography_to_service
Choreography Task T1
Choreography Message CMin_1
Choreography Message CMout_1
Choreography Task Tn
Choreography Message CMin_n
Choreography Message CMout_n
Operation op_1
Message Min_1
Message Mout_1
Operation Relation
1 Message Relation
i Message Relation
Property | Value
Choreography Tasks | Choreography Task T1, ..., Choreography Task Tn
Operations | Operation op_1
Choreography Message | Choreography Message CMin_1
Message | Message Min_1
Choreography Message | Choreography Message CMin_i
Message | Message Min_1

Figure 8: Adapter model – rule 4

Referring to the sequence in Figure 5.(b), let us suppose that a concrete service has been selected to play the role of the participant B. Let us suppose that the concrete service has an operation op_1 that matches the `ChoreographyTasks` $T1, \ldots, Tn$. The related adapter model in Figure 8, whose type is now `choreography_to_service`, has an `OperationRelation` that associates the `ChoreographyTasks` $T1, \ldots, Tn$ to op_1. Note that only the task Tn can have a return message, otherwise it would be practically impossible to generate an adapter. Rule 4 follows.

[Rule 4] Aggregator choreography_to_service – An Aggregator producing the Min_1 out of the messages $CMin\_1, \ldots, CMin\_i$ is generated if the `OperationRelation` has $1 < i \leq n$ `MessageRelations` that involve the initiating messages $CMin\_1, \ldots, CMin\_i$ of $T1, \ldots, Ti$, respectively, and the input message Min_1 of the `Operation` op_1.

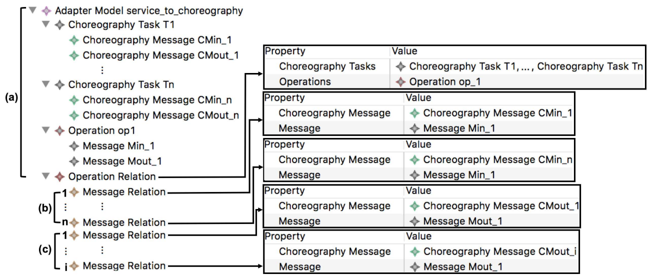

Adapter Model service_to_choreography
Choreography Task T1
Choreography Message CMin_1
Choreography Message CMout_1
Choreography Task Tn
Choreography Message CMin_n
Choreography Message CMout_n
Operation op1
Message Min_1
Message Mout_1
Operation Relation
1 Message Relation
n Message Relation
1 Message Relation
i Message Relation
(a)
(b)
(c)
Property | Value
Choreography Tasks | Choreography Task T1, ..., Choreography Task Tn
Operations | Operation op_1
Choreography Message | Choreography Message CMin_1
Message | Message Min_1
Choreography Message | Choreography Message CMin_n
Message | Message Min_1
Choreography Message | Choreography Message CMout_1
Message | Message Mout_1
Choreography Message | Choreography Message CMout_i
Message | Message Mout_1

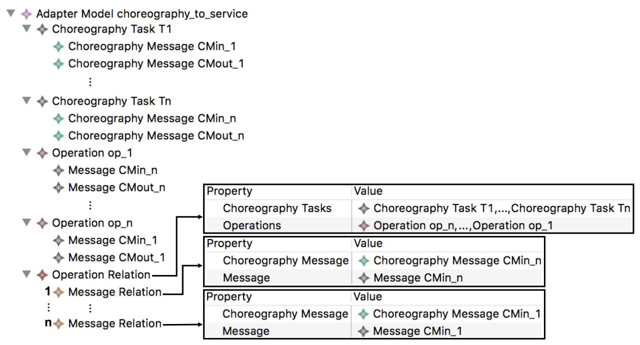
Figure 9: Adapter model – rules 5 and 6

Consider the sequence in Figure 5.(c) and focus on the participant A. The concrete service selected for playing the role of A has an operation op_1 that matches the choreography tasks $T1, \ldots, Tn$. The related `service_to_choreography` adapter model, containing the `OperationRelation` that associates $T1, \ldots, Tn$ to op_1, is shown in Figure 9. The description of rules 5 and 6 follows.

[Rule 5] Splitter service_to_choreography – Referring to Figure 9.(b), a Splitter producing the messages $CMin\_1, \ldots, CMin\_n$ out of the message Min_1 is generated if the `OperationRelation` has n `MessageRelations` that involve the initiating messages $CMin\_1, \ldots, CMin\_n$ and the input message Min_1 of the `Operation` op_1.

[Rule 6] Aggregator service_to_choreography – Referring to Figure 9.(c), an Aggregator is generated if the

OperationRelation has $1 < i \leq n$ MessageRelations that involve the return messages $CMout\_1, \dots, CMout\_i$ of $T1, \dots, Ti$, respectively, and the output message $Mout\_1$ of the Operation op_1. The Aggregator receives as input the messages $CMout\_1, \dots, CMout\_i$ and combine them into $Mout\_1$.



Property	Value
Choreography Tasks	Choreography Task T1,...,Choreography Task Tn
Operations	Operation op_n,...,Operation op_1

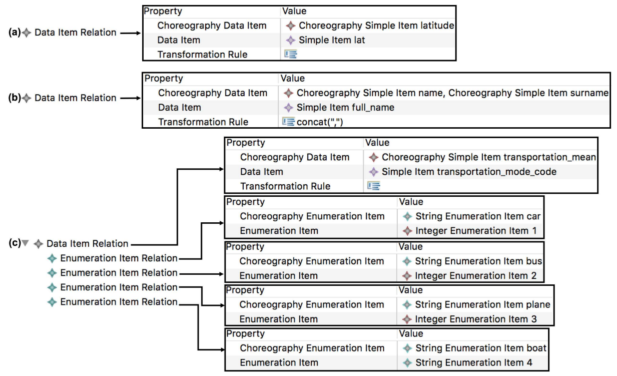
Property	Value
Choreography Message	Choreography Message CMin_n
Message	Message CMin_n

Property	Value
Choreography Message	Choreography Message CMin_1
Message	Message CMin_1

Figure 10: Adapter model – rule 7

Still considering Figure 5.(c), and focussing on the participant B, let us assume that a concrete service playing the role of B has the operations $op\_1, \dots, op\_n$ matching the tasks $T1, \dots, Tn$. Figure 10 shows the related choreography_to_service adapter model that associates $T1, \dots, Tn$ to $op\_1, \dots, op\_n$, respectively. The same holds when the concrete service plays the role of the participant A.

[Rule 7] Resequencer choreography_to_service and service_to_choreography – a Resequencer is generated if the OperationRelation has MessageRelations involving the initiating messages $CMin\_1, \dots, CMin\_n$ of $T1, \dots, Tn$, and the input messages $CMin\_1, \dots, CMin\_n$ of $op\_1, \dots, op\_n$. The generated Resequencer receives as input the messages $CMin\_1, \dots, CMin\_n$ and re-orders them into a permutation (e.g., $CMin\_n, \dots, CMin\_1$ as shown in Figure 10).


(a) Data Item Relation
(b) Data Item Relation
(c) Data Item Relation
Enumeration Item Relation
Enumeration Item Relation
Enumeration Item Relation
Enumeration Item Relation

Property	Value
Choreography Data Item	Choreography Simple Item latitude
Data Item	Simple Item lat
Transformation Rule	

Property	Value
Choreography Data Item	Choreography Simple Item name, Choreography Simple Item surname
Data Item	Simple Item full_name
Transformation Rule	concat(",")

Property	Value
Choreography Data Item	Choreography Simple Item transportation_mean
Data Item	Simple Item transportation_mode_code
Transformation Rule	

Property	Value
Choreography Enumeration Item	String Enumeration Item car
Enumeration Item	Integer Enumeration Item 1

Property	Value
Choreography Enumeration Item	String Enumeration Item bus
Enumeration Item	Integer Enumeration Item 2

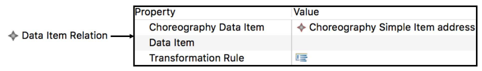
Property	Value
Choreography Enumeration Item	String Enumeration Item plane
Enumeration Item	Integer Enumeration Item 3

Property	Value
Choreography Enumeration Item	String Enumeration Item boat
Enumeration Item	String Enumeration Item 4

Figure 11: Adapter model – rule 8

Rules 8 and 9 concern data-driven adaptation. They can be applied to the data items of the messages involved in a single message relation. Thus, given a MessageRelation between one ChoreographyMessage CM and one Message M, the descriptions of the rule 8 and 9 follow.

[Rule 8] Message Translator – a Message Translator is generated by considering each DataItemRelation, between the ChoregraphyDataItems of CM and the DataItems of M. The conversion performed by the Message Translator can involve either the plain correspondence between the data items of CM to the ones of M (see Figure 11.(a)), or their transformation applying the transformationRules specified in the DataItemRelations (see Figure 11.(b)). Moreover, in case of a DataItemRelation, involving a ChoreographySimpleItem with ChoreographyEnumerationItems and a SimpleItem with EnumerationItems, the conversion is realized by means of the related EnumerationItemRelations (see Figure 11.(c)).



Property	Value
Choreography Data Item	Choreography Simple Item address
Data Item	
Transformation Rule	

Figure 12: Adapter model – rule 9

[Rule 9] Content Filter – As shown in the example in Figure 12, Content Filter is generated by considering each DataItemRelation involving one ChoregraphyDataItem of CM and no DataItem of M, or viceversa. The generated Content Filter removes the unmapped data items, i.e., the ones not involved in any DataItemRelation.

5 ADAPTERS GENERATION AT WORK

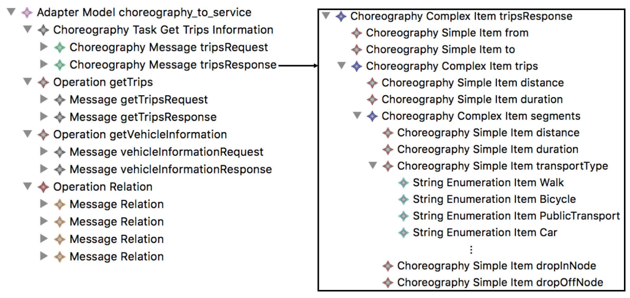
This section describes how the Adapters generation phase of the synthesis process (see Figure 1) realizes a set of possible adaptations by exploiting the related adapter model.

With the purpose of showing the adapters generation at work, we briefly recall some characteristics of the case study that serves as basis for the running example used in the remainder of the section. As described in Section 3, the main scope of the SMT scenario is to realize a CTAS through the cooperation of several content and service providers, organizations and authorities. To this extent, the key issue is to deal with the heterogeneity of the involved concrete service data types and to adapt them to the choreography message data types format. Regarding the smart mobility information, one of the involved participant is the Journey Planner participant. It is in charge of providing trips information to the Mobility Information Planner. This is accomplished through the choreography task Get Trips Information, whose initiating participant, Mobility Information Planner, starts the interaction by sending the message tripsRequest to the receiving participant Journey Planner. Then, the latter replies with the tripsResponse message.

During the selection activity of the synthesis process, the Service Inventory is queried in order to select a concrete service capable of playing the role of the Journey Planner participant. The inventory contains a reference to an external third-party service whose interface does not perfectly match the abstract interface specified for Journey Planner, i.e., the messages specified for it in the BPMN2 choreography diagram. An Adapter is needed. Thus, upon selecting the concrete third-party service available in the inventory, the user is guided through a sequence of steps that lead to the definition of an adapter model conforming to the metamodel in Figure 4. The defined model is then used to automatically generate the code of an adapter capable of bridging the gap between the abstract interface of Journey Planner participant and the concrete one of the third-party service. Moreover, as said above, the Journey Planner is the receiving participant of the considered choreography task. Thus, the adapter is needed to mediate the interaction CD-to-Service.

In the following, the messages related to the choreography task `Get Trips Information` are used to describe the adapters generation at work. For the sake of clarity, we only consider some mapping relations between the choreography return message `tripsResponse` and the response messages of the selected third-party service. As stated above, the interaction prescribed by this choreography task is required to provide trip information to the Mobility Information Planner. Thus, the mapping between the messages concerns the trip information representation. Specifically, the response messages of the selected third-party service has more details than the related return message in the choreography. For example, the selected third-party service provides map information that is not used by the choreography. In addition, the transport type is encoded as an integer value, whereas in the choreography message it is represented with the name (a string) of the transportation means. Furthermore, the starting point of a trip is represented by the data items name, latitude, and longitude, whereas in the choreography only the starting point name is needed. Finally, in case of public transport, drop-in and drop-off bus stop names are represented through different data items with respect to the ones required in the choreography.

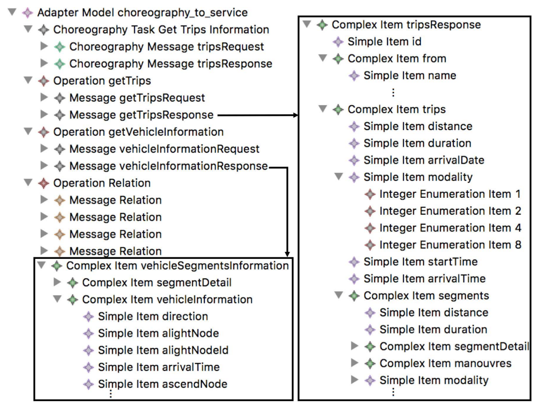
Figures 13, 14 and 15 depict an excerpt of the resulting adapter model. In particular, the right-hand side of Figure 13 shows the detailed definition of the choreography message `tripsResponse`. Regarding this message, we consider the message data items `from`, `dropInNode`, `dropOffNode`, and `transportType`, all of type string. Indeed, the latter enumerates the values $\{Walk, Bicycle, PublicTransport, Car\}$. Referring to the left-hand side of metamodel in Figure 4, the concepts instantiated are `ChoreographySimpleItem`, `StringEnumerationItem`, and hence `ChoreographyEnumerationItem`.


```
▼ Adapter Model choreography_to_service
  ▼ Choreography Task Get Trips Information
    ▶ Choreography Message tripsRequest
    ▶ Choreography Message tripsResponse
  ▼ Operation getTrips
    ▶ Message getTripsRequest
    ▶ Message getTripsResponse
  ▼ Operation getVehicleInformation
    ▶ Message vehicleInformationRequest
    ▶ Message vehicleInformationResponse
  ▼ Operation Relation
    ▶ Message Relation
    ▶ Message Relation
    ▶ Message Relation
    ▶ Message Relation

▼ Choreography Complex Item tripsResponse
    Choreography Simple Item from
    Choreography Simple Item to
  ▼ Choreography Complex Item trips
      Choreography Simple Item distance
      Choreography Simple Item duration
    ▼ Choreography Complex Item segments
        Choreography Simple Item distance
        Choreography Simple Item duration
      ▼ Choreography Simple Item transportType
          String Enumeration Item Walk
          String Enumeration Item Bicycle
          String Enumeration Item PublicTransport
          String Enumeration Item Car
          ⋮
        Choreography Simple Item dropInNode
        Choreography Simple Item dropOffNode
```

Figure 13: Choreography return message

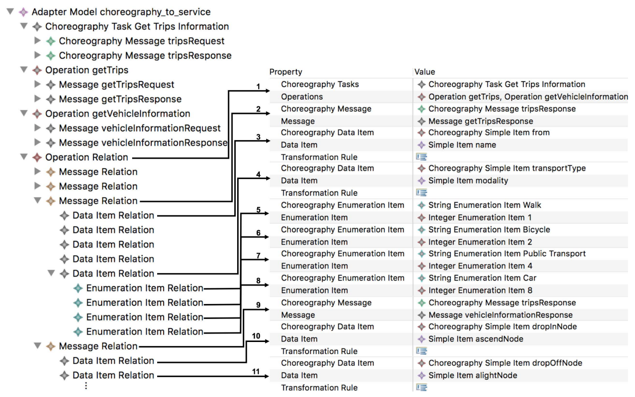
The concrete service selected to play the Journey Planner role has an interface with two operations: `getTrips` and `getVehicleInformation`. The former involves the messages `getTripRequest` and `getTripsResponse`; the latter involves the messages `vehicleInformationRequest` and `vehicleInformationResponse`. We focus on the messages `getTripsResponse` and `vehicleInformationResponse`, whose details are framed in Figure 14. Regarding the message `getTripsResponse`, we consider the data items: `name` of type string, a child of the complex type `from`; `modality` of type integer, a child of the complex type `trips`, whose value is restricted to the set $\{1, 2, 4, 8\}$. Concerning the message `vehicleInformationResponse` the data items of interest are `alightNode` and `ascendNode`, children of the complex type


```
▼ Adapter Model choreography_to_service
  ▼ Choreography Task Get Trips Information
    ▶ Choreography Message tripsRequest
    ▶ Choreography Message tripsResponse
  ▼ Operation getTrips
    ▶ Message getTripsRequest
    ▶ Message getTripsResponse
  ▼ Operation getVehicleInformation
    ▶ Message vehicleInformationRequest
    ▶ Message vehicleInformationResponse
  ▼ Operation Relation
    ▶ Message Relation
    ▶ Message Relation
    ▶ Message Relation
    ▶ Message Relation
  ▼ Complex Item vehicleSegmentsInformation
    ▶ Complex Item segmentDetail
    ▼ Complex Item vehicleInformation
        Simple Item direction
        Simple Item alightNode
        Simple Item alightNodeId
        Simple Item arrivalTime
        Simple Item ascendNode
        ⋮

▼ Complex Item tripsResponse
    Simple Item id
  ▼ Complex Item from
      Simple Item name
      ⋮
  ▼ Complex Item trips
      Simple Item distance
      Simple Item duration
      Simple Item arrivalDate
    ▼ Simple Item modality
        Integer Enumeration Item 1
        Integer Enumeration Item 2
        Integer Enumeration Item 4
        Integer Enumeration Item 8
      Simple Item startTime
      Simple Item arrivalTime
    ▼ Complex Item segments
        Simple Item distance
        Simple Item duration
      ▶ Complex Item segmentDetail
      ▶ Complex Item manouvres
      ▶ Simple Item modality
        ⋮
```

Figure 14: Output message

`vehicleInformation`, both of type string. Referring to the right-hand side of the metamodel in Figure 4, the involved concepts are `ComplexItem`, `SimpleItem`, `IntegerEnumerationItem`, and hence `EnumerationItem`.


```
▼ Adapter Model choreography_to_service
  ▼ Choreography Task Get Trips Information
    ▶ Choreography Message tripsRequest
    ▶ Choreography Message tripsResponse
  ▼ Operation getTrips
    ▶ Message getTripsRequest
    ▶ Message getTripsResponse
  ▼ Operation getVehicleInformation
    ▶ Message vehicleInformationRequest
    ▶ Message vehicleInformationResponse
  ▼ Operation Relation
    ▶ Message Relation
    ▶ Message Relation
    ▼ Message Relation
        Data Item Relation
        Data Item Relation
        Data Item Relation
        Data Item Relation
      ▼ Data Item Relation
          Enumeration Item Relation
          Enumeration Item Relation
          Enumeration Item Relation
          Enumeration Item Relation
    ▼ Message Relation
        Data Item Relation
        Data Item Relation
        ⋮
```

Property	Value
Choreography Tasks	Choreography Task Get Trips Information
Operations	Operation getTrips, Operation getVehicleInformation
Choreography Message	Choreography Message tripsResponse
Message	Message getTripsResponse
Choreography Data Item	Choreography Simple Item from
Data Item	Simple Item name
Transformation Rule	
Choreography Data Item	Choreography Simple Item transportType
Data Item	Simple Item modality
Transformation Rule	
Choreography Enumeration Item	String Enumeration Item Walk
Enumeration Item	Integer Enumeration Item 1
Choreography Enumeration Item	String Enumeration Item Bicycle
Enumeration Item	Integer Enumeration Item 2
Choreography Enumeration Item	String Enumeration Item Public Transport
Enumeration Item	Integer Enumeration Item 4
Choreography Enumeration Item	String Enumeration Item Car
Enumeration Item	Integer Enumeration Item 8
Choreography Message	Choreography Message tripsResponse
Message	Message vehicleInformationResponse
Choreography Data Item	Choreography Simple Item dropInNode
Data Item	Simple Item ascendNode
Transformation Rule	
Choreography Data Item	Choreography Simple Item dropOffNode
Data Item	Simple Item alightNode
Transformation Rule	

Figure 15: Adapter model mappings

Figure 15 shows the mappings between the choreography message `tripsResponse` and the concrete messages `getTripsResponse` and `vehicleInformationResponse`. First, the choreography task `Get Trips Information` is associated with the operations `getTrips` and `getVehicleInformation` (relation **1**). Then, the choreography message `tripsResponse` is related to both the message `getTripsResponse` (relation **2**) and the message `vehicleInformationResponse` (relation **9**) (see `MessageRelation` in the metamodel).

Regarding the message data type relations (see `DataItemRelation` in the metamodel), we only describe the most representative mappings with respect to the generation of the Message Translator and the Content Filter required for adapting the task `Get Trips Information` considered in our example. In particular, the Message Translator is synthesised by considering the following pairs of choreography simple types and concrete service simple types, and hence the pairs {`ChoreographySimpleItem`, `SimpleItem`} in the metamodel: {`dropInNode`, `ascendNode`} (relation **10**) and {`dropOffNode`, `alightNode`} (relation **11**) that concern the message relation **9** (`tripsResponse`, `vehicleInformationResponse`); {`from`,

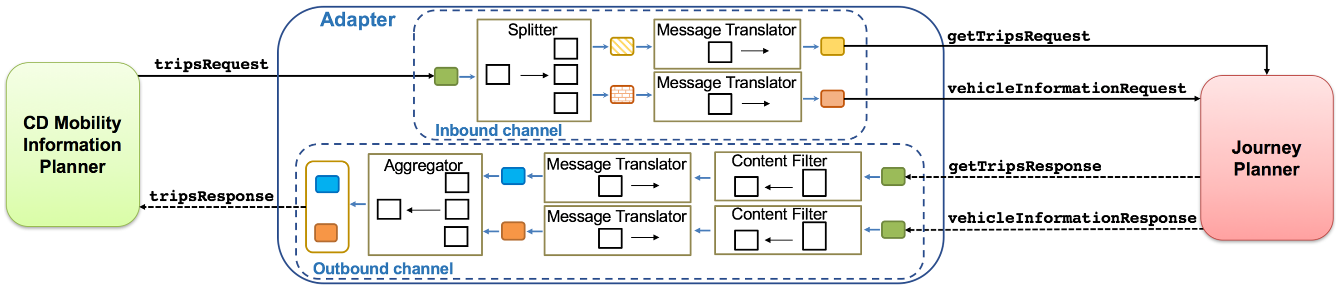


Figure 16: Journey Planner Adapter

name} (relation **3**) and {transportType, modality} (relation **4**) that concern the message relation **2** (tripsResponse, getTripsResponse). The last mapping {transportType, modality} is further detailed by means of the enumeration type relations (EnumerationItemRelation in the metamodel): {Walk, 1} (relation **5**), {Bicycle, 2} (relation **6**), {Public Transport, 4} (relation **7**), {Car, and 8} (relation **8**).

Starting from the adapter model and applying the adapters generation rules defined in Section 4.2, the adapter generation activity of the synthesis process (Section 2) generates the adapter. Figure 16 shows the adapter that, according to the architectural style in Figure 2, is interposed between the CD and the related concrete service Journey Planner. Via the inbound channel, the resulting adapter receives as input the message tripsRequest that, after being split, is translated into the messages getTripsRequest and vehicleInformationRequest. Upon receiving the messages getTripsResponse and vehicleInformationResponse through the outbound channel, the adapter creates the message tripsResponse. For the sake of space, in the following, we focus on describing the outbound channel only.

The adapter model has two MessageRelations associating the message getTripsResponse and vehicleInformationResponse to the same choreography message tripsResponse, together with their data items. This results in both Flow-Driven and Data-driven adaptation.

Data-driven adaptation is obtained through Rule 8 and Rule 9. In particular, by applying Rule 9, two Content Filters are generated to delete the data items of the messages getTripsResponse and vehicleInformationResponse that are not involved in any of the message data type relations. Then, by applying Rule 8, two Message Translators are realised in order to convert the remaining data items to the data items of the message tripsResponse. Finally, by applying Rule 2, an Aggregator is derived. It takes as input the messages produced by the translators and combines them into the message tripsResponse.

6 RELATED WORK

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 [13, 21].

Spitznagel and Garlan propose an approach to formally specify adapter wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches [28].

Passerone et al. 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 [25]. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adapter, by delegating to the user the non-trivial task of specifying that. Our validation phase, automatically achieve deadlock freeness in the choreography specification.

Recently, Bennaceur and Issarny presented an approach that, exploiting ontology reasoning and constraint programming, allows for automatically inferring mappings between components interfaces [11]. Importantly, these mappings guarantee semantic compatibility between the operations and data. Although valuable and powerful, this approach does not account for development effort whose aim is to bring the adoption of choreographies to the development practices currently adopted by IT companies.

In [14] the authors present a generic schema match system called COMA, which provides an extensible library of simple and hybrid match algorithms and supports a powerful framework for combining match results. This framework can be used for systematically evaluate different aspects of match processing, match direction, match candidate selection, and computation of combined similarity, and different matcher usages.

Paolucci et al. propose a base algorithm [24] for semantic matching between service advertisements and service requests based on DAML-S, a DAML-based language for service description. The algorithm proposed differentiate between four degrees of matching and can be used for automatic dynamic discovery, selection and inter-operation of web services.

In [22] the authors discuss an extensions of the Jolie orchestration language [23], namely JoRBA framework that allows to develop dynamically adaptable service oriented applications. In [26] the AIOCJ choreography language is presented, where leveraging on adaptability features of JoRBA, allows for developing adaptable choreographies.

In [16], the authors propose an approach to enforce synchronizability and realizability of a choreography. The approach is able to automatically generate monitors, which act as local controllers interacting with their peers and the rest of the system in order to make the peers respect the choreography specification. Our notion of CD is “similar” to the notion of monitor used in [16]. However, the two synthesis methods are different.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we presented the model-based approach used within the CHOReVOLUTION project to achieve choreography adaptation. The approach is that of automatically generating adapters by combining different EIPs according to a notion of protocol mediation and data similarity.

Our approach supports patterns belonging to the Message Routing and Message Transformation EIP classes [18]. The former patterns allow to decouple a message source from the ultimate destination of the message according to message routing policies. The latter patterns offer a general solution to possible differences in the data format of the exchanged messages enabling a finer form of adaptation concerning mismatches at the level of the semantics of the exchanged messages.

Furthermore, the ability to deal with these patterns allows to support two types of choreography adaptation: (i) *Flow-driven adaptation* that operates at the granularity of messages adapting the flow of messages among participants, and (ii) *Data-driven adaptation* that operates at the granularity of the data items constituting the message adapting the content of messages among participants.

An explanatory example, taken from the Smart and Mobility Tourism use case of the CHOReVOLUTION project, has been used to show the Message Translator pattern and Content Filter pattern. They allowed to mediate the interaction with a third-party service, which uses a data format different from the one defined for a choreography message.

As future work, we plan to add support to the automated identification of interoperability issues in order to automatically infer data mappings between different messages, and then automatically derive the adapter model. In this direction, we can exploit Strawberry tool [12] that through a testing phase, based on an ad-hoc oracle, is able to infer the semantic correlation between two messages. An alternative way to check message semantic correlation would be to exploit ontological information in a way similar to what is described in [19].

8 ACKNOWLEDGMENTS

This research work has been supported by (i) the EU H2020 Programme under grant agreement number 644178 (project CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), (ii) the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies), and (iii) the GAUSS national research project funded by the MIUR under the PRIN 2015 program (Contract 2015KWREMX).

REFERENCES

[1] Marco Autili, Amleto Di Salle, Alexander Perucci, and Massimo Tivoli. 2015. On the Automated Synthesis of Enterprise Integration Patterns to Adapt Choreography-based Distributed Systems. In *Proc. of 14th Coordination Languages and Self-Adaptive Systems (FOCLASA) (EPTCS)*. 33–47.

[2] Marco Autili, Paola Inverardi, Filippo Mignosi, Romina Spalazzese, and Massimo Tivoli. 2015. Automated Synthesis of Application-Layer Connectors from Automata-Based Specifications. In *9th Int. Conf. on Language and Automata Theory and Applications LATA*. 3–24.

[3] Marco Autili, Paola Inverardi, and Massimo Tivoli. 2014. CHOREOS: Large scale choreographies for the future internet. In *2014 Software Evolution Week - IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE*. 391–394.

[4] M. Autili, P. Inverardi, and M. Tivoli. 2015. Automated Synthesis of Service Choreographies. *IEEE Software* 32, 1 (Jan 2015), 50–57.

[5] Marco Autili, Leonardo Mostarda, Alfredo Navarra, and Massimo Tivoli. 2008. Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems. *Journal of Systems and Software* 81, 12 (2008), 2210–2236.

[6] Marco Autili, Davide Ruscio, Amleto Di Salle, Paola Inverardi, and Massimo Tivoli. 2013. A Model-Based Synthesis Process for Choreography Realizability Enforcement. In *FASE*. LNCS, Vol. 7793.

[7] Marco Autili, Amleto Di Salle, and Massimo Tivoli. 2013. Synthesis of Resilient Choreographies. In *5th Int. Workshop on Software Engineering for Resilient Systems SERENE*. 94–108.

[8] Marco Autili and Massimo Tivoli. 2015. Distributed Enforcement of Service Choreographies. In *Proc. 13th Int. Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 6th September 2014*. 18–35.

[9] Samik Basu and Tevfik Bultan. 2011. Choreography conformance via synchronizability. In *Proc. of WWW '11*.

[10] Samik Basu, Tevfik Bultan, and Meriem Ouederni. 2012. Deciding choreography realizability. In *POPL*. ACM.

[11] Amel Bennaceur and Valérie Issarny. 2015. Automated Synthesis of Mediators to Support Component Interoperability. *IEEE Trans. Software Eng.* 41, 3 (2015), 221–240.

[12] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. 2009. Automatic Synthesis of Behavior Protocols for Composable Web-services. In *Proc. of the the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*. 141–150.

[13] Kenneth L. Calvert and Simon S. Lam. 1990. Formal Methods for Protocol Conversion. *IEEE Journal on Selected Areas in Communications* 8, 1 (1990).

[14] Hong Hai Do and Erhard Rahm. 2002. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. of 28th Int. Conf. on Very Large Data Bases*. 610–621.

[15] Matthias Güdemann, Pascal Poizat, Gwen Salaün, and Lina Ye. 2016. VerChor: A Framework for the Design and Verification of Choreographies. *IEEE Transaction on Services Computing* 9, 4 (2016), 647–660.

[16] Matthias Güdemann, Gwen Salaün, and Meriem Ouederni. 2012. Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In *Automated Technology for Verification and Analysis*, Supratik Chakraborty and Madhavan Mukund (Eds.). 238–253.

[17] Sylvain Hallé and Tevfik Bultan. 2010. Realizability analysis for message-based interactions using shared-state projections. In *Proc. of the eighteenth ACM SIGSOFT Int. symposium on Foundations of software engineering (FSE '10)*. 27–36.

[18] Gregor Hohpe and Bobby Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions - Fiftheenth printing 2011*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[19] Paola Inverardi and Massimo Tivoli. 2013. Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns. In *Proc. of the 2013 Int. Conf. on Software Engineering (ICSE '13)*. 3–12.

[20] Raman Kazhamiakin and Marco Pistore. 2006. Analysis of Realizability Conditions for Web Service Choreographies. In *26th IFIP WG FORTE*. 61–76.

[21] Simon S. Lam. 1988. Correction to "Protocol Conversion". *IEEE Trans. Software Eng.* 14, 9 (1988).

[22] Ivan Lanese, Fabrizio Montesi, and Gianluigi Zavattaro. 2015. The Evolution of Jolie. In *Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering (LNCS)*. 506–521.

[23] Fabrizio Montesi, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. 2007. JOLIE: a Java Orchestration Language Interpreter Engine. *Electronic Notes in Theoretical Computer Science* 181 (2007), 19 – 33.

[24] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. 2002. Semantic Matching of Web Services Capabilities. In *The Semantic Web - ISWC 2002, 1st Int. Semantic Web Conf., Sardinia, Italy, June 9-12, 2002, Proceedings*. 333–347.

[25] Roberto Passerone, Luca de Alfaro, Thomas A. Henzinger, and Alberto L. Sangiovanni-Vincentelli. 2002. Convertibility verification and converter synthesis: two faces of the same coin. In *ICCAD*.

[26] Mila Dalla Preda, Saverio Giallorenzo, Ivan Lanese, Jacopo Mauro, and Maurizio Gabbrielli. 2014. AIOCJ: A Choreographic Framework for Safe Adaptive Distributed Applications. *CoRR* abs/1407.0975 (2014).

[27] Gwen Salaün, Tevfik Bultan, and Nima Roohi. 2012. Realizability of Choreographies Using Process Algebra Encodings. *IEEE Trans. Services Computing* 5, 3 (2012), 290–304.

[28] Bridget Spitznagel and David Garlan. 2003. A Compositional Formalization of Connector Wrappers. In *ICSE*.