On the model-driven synthesis of adaptable choreographies

Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, Massimo Tivoli
University of L'Aquila, Italy
{marco.autili,amleto.disalle,francesco.gallo,massimo.tivoli}@univaq.it
claudio.pompilio@graduate.univaq.it

ABSTRACT

Service choreographies represent a powerful and flexible approach to compose software services in a distributed way. A key enabler for the 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 is needed. Adapters are used to bind concrete services to (abstract) choreography roles by solving possible protocol mismatches. In this paper we focus on the synthesis of adapters, each of them realized as a suitable composition of adaptation primitives. We show our method at work on a use case in the domain of Smart Mobility & Tourism.

1 INTRODUCTION

Service choreographies represent a powerful and flexible approach to compose software services in a distributed way. Many theoretic approaches have been proposed [9, 14, 15, 17, 23], 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 realize service choreographies [1–7] by possibly reusing third-party services. 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 v.2.0[2] (BPMN2), which introduced *Choreography Diagrams* to offer choreography modeling constructs. 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 services 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.*

To address this problem, beyond coordination, adaptation must be achieved automatically. To this extent, in the CHOReVOLUTION project we have developed a process to synthesize a set of software entities that interposed among the concrete services enforce the collaboration prescribed by the choreography specification. In particular, Adapters solve possible protocol mismatches between services and choreography roles. In [8], we defined an approach to the automatic synthesis of service Adapters by employing Enterprise Integration Patterns [16] (EIPs). EIPs serve as adaptation primitives to realize the required adaptation logic. In this paper, we describe how the adaptation logic of an Adapter is obtained as a chain of adapter components implementing the related EIPs.

The paper is structured as follow. Section 2 provides the background. Section 3 focuses on the approach to choreography adaptation, and Section 4 describes it at work on an explanatory example. Related work is discussed in Section 5, and conclusions are given in Section 6.

2 BACKGROUND

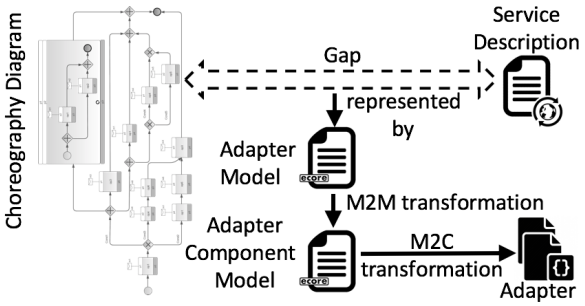

Figure 1: Background scenario

Service choreography is a decentralized approach, which provides a loose way to compose service by specifying the participants (i.e., roles) and the (message-based) interaction protocol between them. As already introduced, a role in a BPMN2 choreography diagram models the expected behaviour (e.g., the expected protocol) that a concrete participant service should match in order to be able to play the role of the choreography. In other words, roles characterize the behavioural boundaries that services playing them must obey. Within these boundaries, the behaviour of the concrete participant services can be adapted. This means that there is a gap between the abstract protocol of the choreography participant roles and the protocol of the concrete services, as shown in Figure 1. This gap consists of mismatches between protocols by means of (possibly complex) data mappings over both operation names and I/O messages, and their flows. In [8], we represented this matching through an Adapter metamodel. It considers the subset of EIPs [16] that belong to the classes of Message Routing and Message Transformations Patterns. In particular, Message Routing EIPs are applied to adapt sequences of messages, and hence realizing **Flow-driven adaptation**. Instead, Message Transformation EIPs are applied to

[1] www.choreos.eu – www.chorevolution.eu
[2] http://www.omg.org/spec/BPMN/2.0.2/

ModComp'18, Oct. 14-16, 2018, Copenhagen, Denmark

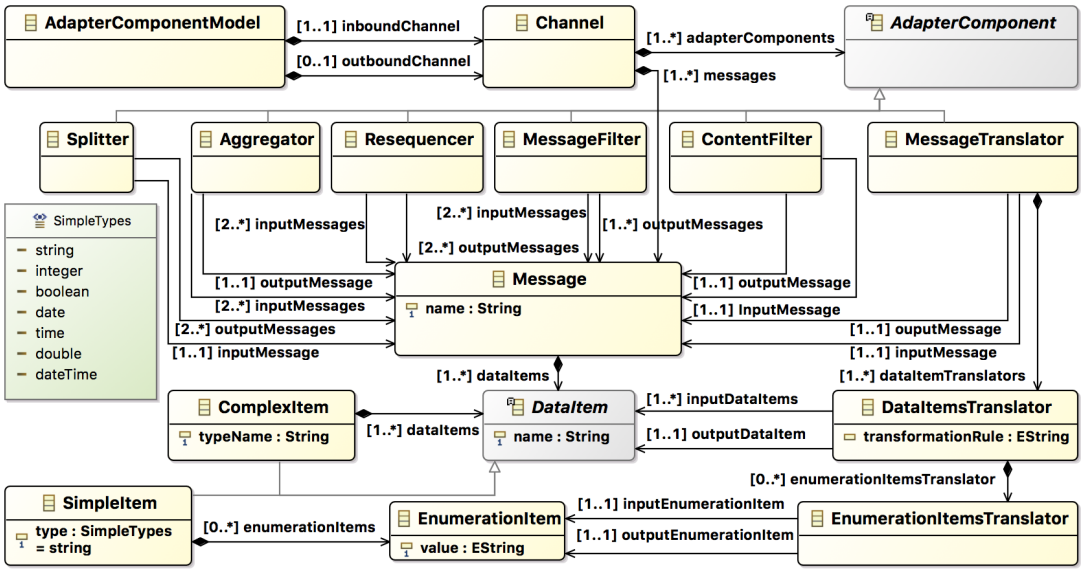

Figure 2: Adapter Component Metamodel

adapt the content of a single exchanged message, and hence realizing **Data-driven adaptation**. Moreover, in [8], we defined a set of rules corresponding to the EIPs adaptation primitives underlying the synthesis of Adapters.

To better understand the problem that is being solved, we introduce a case study from the CHOReVOLUTION project named Smart Mobility and Tourism (SMT). An excerpt of the SMT case study is used as running example in the remainder of this paper. 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. Concerning the smart mobility information, one of the involved participant is the Parking participant. It provides parking information to the Mobility Information Planner (MIP). The choreography prescribes that this information is gathered through a single message exchange. Moreover, it contains both the data of the nearest parking according to the user position and the directions to reach it.

The concrete service that plays the role of the Parking participant has an interface that does not perfectly match the related abstract interface in the choreography. In particular, it provides an operation to get the nearest parking in a specific area and an operation to calculates the related route. Furthermore, several data items are encoded in different formats and the response messages of concrete service have more details than the related choreography message. An Adapter model represents the gap between the abstract protocol of the MIP choreography participant and the protocol of the concrete service playing the role of the Parking participant.

In this paper, we define a metamodel, namely the Adapter Component metamodel, that represents the structure of an adapter as a chain of adapter components implementing the considered EIPs. Furthermore, we specify a model-to-model transformation from an adapter model to an adapter component model. The adapter component metamodel is a platform-independent representation. Thus, several technologies, such as Spring Integration[3], Apache Camel[4], can be supported by defining the related model-to-code transformation. In CHOReVOLUTION we adopted Spring Integration framework as target implementation.

3 APPROACH

In this section we describe the proposed approach. First, we present the adapter component metamodel. Then, we discuss the model-to-model transformation to obtain an adapter component model from an adapter model.

3.1 Adapter Component Metamodel

The adapter component metamodel in Figure 2 represents the structure of an adapter. It contains an inbound channel and an outbound channel, if needed. Each channel is a chain of adapter components implementing the considered EIPs. In particular, a Splitter has an input message and two or more messages as output, whereas the opposite holds for the Aggregator. A Resequencer has two or more messages both as input and output. The desired order of the output messages is specified by a a property of the Resequencer (not shown in the figure). A Message Filter has two or more input messages and one or more messages as output. The Content Filter and Message Translator operate at the granularity of the data items constituting the message. Thus, they have a single message both as input and output. The Content Filter removes the data items should not be present in the output message, whereas the Message Translator transforms data items to convert a message from one format to another one. This conversion is defined by associating one or more input data items with one output data item, possibly

[3]https://spring.io/projects/spring-integration
[4]https://camel.apache.org/

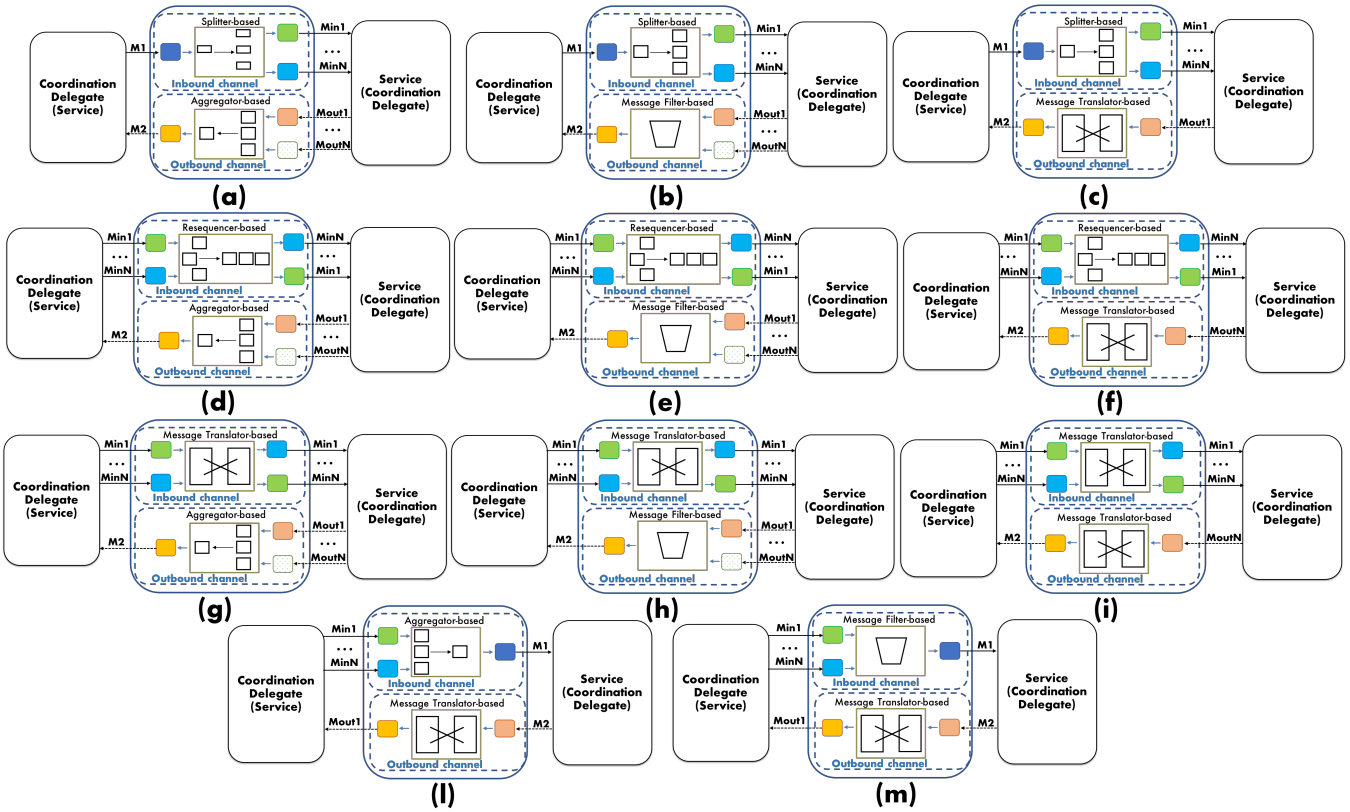

Figure 3: Adaptation scenarios

with a transformation rule. Moreover, if the involved data items are enumerated, the conversion can be further detailed by associating the enumerated values.

3.2 Adapter Component Transformation

This section describes the Adapter component transformation. It takes as input an adapter model and generates the related adapter component model. We recall that an adapter model contains the mapping among choreography messages and service messages together with their data items [8].

For the sake of space, we provide a high-level description of the main steps of the transformation algorithm. First, it parses the adapter model to detect the required adaptation scenario. Figure 3 reports the possible adaptation scenarios derived from the adapter generation rules defined in [8]. Then, the transformation defines the adapter channels as a chain of adapter components implementing the considered EIPs. In particular, Adapters have inbound channels to adapt request messages and outbound channels to adapt response messages, if present.

As said in Section 1, the CHOReVOLUTION process synthesizes a set of software entities to enforce the collaboration prescribed by the choreography specification. In particular, Adapters are connected with the software entities in charge of coordinating the interactions, called *Coordination Delegates* (CDs). Therefore, Adapters mediate two kinds of interactions: CD-to-Service and Service-to-CD. The adaptation scenarios are symmetric according to the kind of interactions mediated by the adapter. Thus, the adapters have on each side both a coordination delegate and a service.

All the channels of the adapters are labelled with an EIP–based noun. This noun indicates the main EIP of the chain applied according to the behaviour of the adapter channel. In particular, if there is a one–to–many mapping between the request messages, the inbound channel splits the request message (Splitter–based inbound channel in **(a)**, **(b)** and **(c)**). The outbound channel can combine the response messages (Aggregator–based outbound channel in **(a)**). Otherwise, it can filter the response messages (Message Filter–based outbound channel in **(b)**). If there is one response message, the outbound channel can translate it into another format (Message Translator–based outbound channel in **(c)**).

The same applies for the outbound channels in **(d)**, **(e)**, **(f)**, **(g)**, **(h)** and **(i)** associated with the inbound channels that can either re-order the request messages (Resequencer–based inbound channel in **(d)**, **(e)** and **(f)**) or convert them into another format (Message Translator–based inbound channel in **(g)**, **(h)** and **(i)**). These adapters are generated when there is a many–to–many mapping between the request messages.

In case of a many–to–one mapping between the request messages, the inbound channel can either combine (Aggregator–based

inbound channel **(l)**) or filter the request messages (Message Filter–based inbound channel **(m)**), whereas the outbound channel can only transform the response message into another format (Message Translator–based outbound channel in **(l)** and **(m)**).

As said above, the transformation after detecting the required adaptation scenario, it defines the adapter channels as a chain of adapter components implementing the considered EIPs. The following describes the structure of the EIP–based channels. The dashed elements indicate the optional adapter components.

Figure 4 shows the EIPs chain of the Splitter–based channel. In particular, first, a Content Filter removes the data items not mapped by any relation. Then, a Message Translator converts the resulting message into an intermediate message as prescribed by the data items relations. Finally, the intermediate message is split into several messages. The Splitter EIP is applied at the end since there could be data items involved in different relations. This means that splitting the message at the begin into several intermediate messages containing the data items to be translated is not always possible.

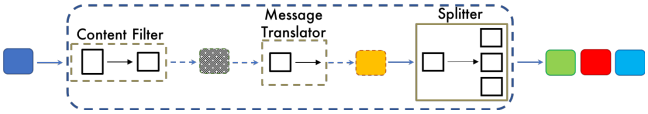

Figure 4: Splitter–based channel

Concerning the Aggregator–based channel, its EIPs chain is illustrated in Figure 5. First, a Message Filter drops the messages not involved in any message relation. Then, for each remaining message, a Content Filter removes the data items not mapped in any relation. After, the intermediate messages are combined by an Aggregator. Finally, a Message Translator converts the resulting message into the format prescribed by the data items relations. The Message Translator is applied after the Aggregator because there could be data items relations involving different messages. Thus, the conversion can be realized only after aggregating the messages into a single message.

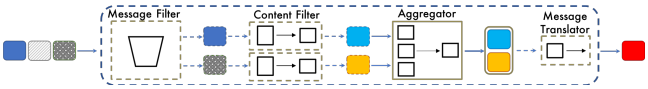

Figure 5: Aggregator–based channel

Figure 6 shows the EIPs chain of the Message Filter–based channel. In particular, first, the messages are filtered. Then, a Content Filter removes the data items of the remaining message not mapped in any relation. Finally, a Message Translator converts the resulting message into the format prescribed by the data items relations.

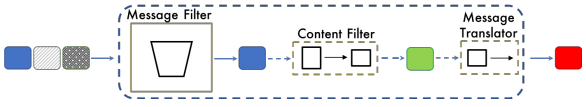

Figure 6: Message Filter–based channel

Regarding the Resequencer–based channel, its chain is illustrated in Figure 7. First, a Content Filter removes the data items not mapped in any relation. Then, for each resulting message, a Message Translator converts it into the format prescribed by the data items relations. Finally, a Resequencer re-orders the messages into the required permutation.

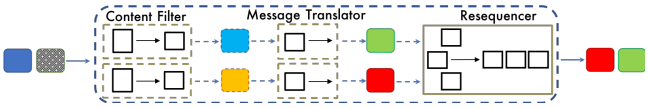

Figure 7: Resequencer–based channel

Figure 8 shows the EIPs chain of the Message Translator–based channel for a single exchanged message. In case the Message Translator–based adapter channel transforms more than one message (Message Translator–based inbound channel in (g), (h) and (i)) this EIPs chain is applied to each exchanged message. In particular, a Content Filter removes the data items not mapped in any relation. Then, a Message Translator converts the resulting message into the format prescribed by the data items relations.

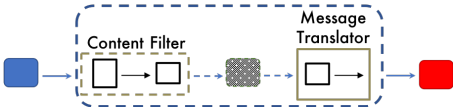

Figure 8: Message Translator–based channel

4 APPROACH AT WORK

This section describes our approach at work on the SMT case study introduced in Section 2. We focus on the interaction between the MIP and the Parking participant to gather parking information. In particular, MIP sends the message `parkingRequest` to the receiving participant Parking. Then, it replies with the message `parkingResponse`. The concrete service that plays the role of the receiving participant Parking has an interface that does not match the related abstract interface in the choreography. Thus, an Adapter that mediates the interaction CD-to-Service is needed. The related adapter model is not reported in this paper, but its messages and data items mappings are used to describe the adapter component transformation at work. Figure 10 shows the resulting adapter components model, whereas Figure 9 shows the related adapter.

In the following we describe the adapter components model by considering the most representative messages and data items relations according to the chains of EIP constituting the inbound and the outbound channels.

The concrete service has an interface with two operations: `getParkingRequest` provides information about the nearest parking in a specific area and `getParkingDirections` calculates the related route. The former involves the messages `getParkingRequest` and `getParkingResponse`; the latter involves the messages `getParkingDirectionsRequest` and `getParkingDirectionsResponse`.

Concerning the request messages, the choreography message `parkingRequest` is mapped to the service messages

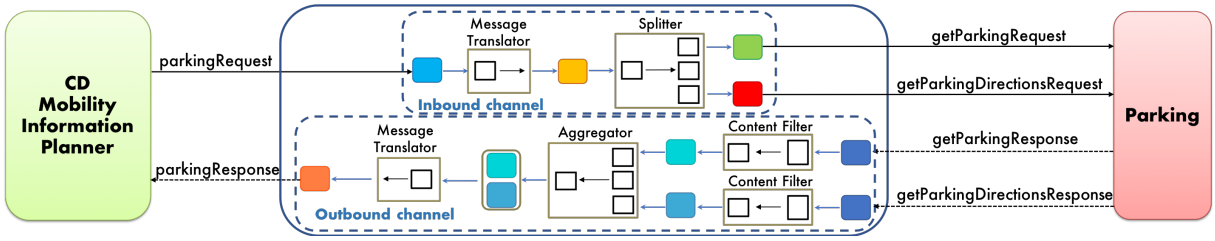

Figure 9: Adapter parking

Figure 10: Adapter parking component model

`getParkingRequest` and `getParkingDirectionsRequest`. This results in a splitter-based inbound channel. Regarding the response messages, the service messages `getParkingResponse` and `getParkingDirectionsResponse` are mapped to the choreography message `getParkingResponse`. This leads to an aggregator-based outbound channel. The transformation identifies the adaptation scenario (a) in Figure 3. Then, it defines the inbound and outbound channels by applying the chain of EIPs specified in the Figures 4 and 5 respectively.

The message `parkingRequest` contains the current position, the upper left and lower down points of the parking area of interest represented by the latitude and longitude coordinates. Moreover, it contains the start and the end time of the parking. The messages `getParkingRequest` and `getParkingDirectionsRequest` contains the starting position and the boundaries of the parking area of interest represented as a string consisting of the latitude and the longitude coordinates. Furthermore, the message `getParkingRequest` contains the parking start and end date.

With respect to the inbound channel (Figure 4), a Content Filter is not needed because all the data items of the request message are involved in some relation. Instead, a Message Translator (relation **1** of Figure 10) is required to convert the data items of the message `parkingRequest` into an intermediate message as prescribed by the data items relations. In particular, it concatenates the coordinates of the current position, upper left and lower down points of the parking area by using the comma as a separator (relation **2** for the upper left point). Furthermore, it converts the start and the end time of the parking into the related date representations by appending them to the current date (relation **3** for the end time). Then, the resulting message is split into the messages `getParkingRequest` and `getParkingDirectionsRequest` (relation **4**).

Regarding the outbound channel (Figure 5), a Message Filter is not needed because all the response messages are involved in some message relation. The message `getParkingDirectionsResponse` contains the name, address and the route from the current position to the parking. All these data items except for the route are not involved in any data item relation so a Content Filter drops them (relation **5**). The message `getParkingResponse` contains the name, latitude, longitude, operator name, type, status, address, price, capacity and the parking rate of the parking. In particular, the type and the status item are represented by an enumeration of integer numbers, whereas in the message `getParkingResponse` they are represented as an enumeration of strings. The capacity and the parking rate items are not involved in any data item relation so they are dropped by a Content Filter (relation **6**). Then, the resulting filtered messages are combined into an intermediate message (relation **7**). Finally, a Message Translator converts this message into the message `parkingResponse` (relation **8**). It copies the data items that do not require a conversion, and it translates the type and the status enumerated integer values into the related enumerated string values. Regarding the type data item (relation **9**), the integers {0, 1, 2, 4} are translated respectively into the strings {unknown, open, multistorey, subterranean} (relations **10**, **11**, **12** and **13**).

5 RELATED WORK

The work described in this paper is related to approaches conceived for providing choreography developers with support for specifying participant adapters and generating their code out of their specification. Thus, in the following, we do not consider on-the-fly adapters generation performed at run time.

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 [11, 18].

With the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [25].

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 [24].

Bennaceur and Issarny presented an approach that, exploiting ontology reasoning and constraint programming, allows for automatically inferring mappings between components interfaces [10]. 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.

Do et al. propose a catalog of criteria for documenting the evaluations of schema matching systems [12]. In particular, the authors discuss various aspects that contribute to the match quality obtained as the result of an evaluation. In [13] the authors present a generic schema match system called COMA, which provides an extensible library of simple and hybrid match algorithms and supports a 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 [21] 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 [19] the authors discuss an extensions of the Jolie orchestration language [20], namely JoRBA framework that allows to develop dynamically adaptable service oriented applications. In [22] the AIOCJ choreography language is presented, where leveraging on adaptability features of JoRBA, allows for developing adaptable choreographies.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented how the model-based approach used within the CHOReVOLUTION project combines different EIPs as adaptation primitives to achieve choreography adaptation. The approach is that of automatically generating a platform-independent representation of the adaptation logic as a chain of adapter components implementing the considered EIPs. In particular, in CHOReVOLUTION we used this representation for generating the related Adapter by using Spring technologies. An explanatory example, taken from the Smart and Mobility Tourism use case of the CHOReVOLUTION project, has been used to show the approach at work.

As future work, we plan to extend the supported implementation technologies of the model-based approach. Furthermore, we will investigate how to support adaptation scenarios involving choreography tasks with multi-instance Participant.

7 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, Amleto Di Salle, and Massimo Tivoli. 2013. Synthesis of Resilient Choreographies. In *SERENE*.
[3] 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.
[4] Marco Autili, Paola Inverardi, and Massimo Tivoli. 2014. CHOREOS: Large scale choreographies for the future internet. In *2014 SW Evolution Week - IEEE Conf. on SW Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE*. 391–394.
[5] M. Autili, P. Inverardi, and M. Tivoli. 2015. Automated Synthesis of Service Choreographies. *Software, IEEE* 32, 1 (Jan 2015), 50–57.
[6] 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.
[7] 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.
[8] Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, and Massimo Tivoli. 2018. Model-driven adaptation of service choreographies. In *Proc. of the 33rd ACM Symposium on Applied Computing, Pau, France, April 09-13*. 1441–1450.
[9] Samik Basu and Tevfik Bultan. 2011. Choreography conformance via synchronizability. In *Proc. of WWW '11*.
[10] Amel Bennaceur and Valérie Issarny. 2015. Automated Synthesis of Mediators to Support Component Interoperability. *IEEE Trans. Soft. Eng.* 41, 3 (2015), 221–240.
[11] Kenneth L. Calvert and Simon S. Lam. 1990. Formal Methods for Protocol Conversion. *IEEE Journal on Selected Areas in Communications* 8, 1 (1990).
[12] Hong Hai Do, Sergey Melnik, and Erhard Rahm. 2002. Comparison of Schema Matching Evaluations. In *Web, Web-Services, and Database Systems, NODe Web and Database-Related Workshops, Erfurt, Germany, October 7-10, 2002*. 221–237.
[13] Hong Hai Do and Erhard Rahm. 2002. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB 2002, Proceedings of 28th International Conf. on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*. 610–621.
[14] 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.
[15] Sylvain Hallé and Tevfik Bultan. 2010. Realizability analysis for message-based interactions using shared-state projections. In *Proc. of the 18th ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10)*. 27–36.
[16] 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.
[17] Raman Kazhamiakin and Marco Pistore. 2006. Analysis of Realizability Conditions for Web Service Choreographies. In *26th IFIP WG FORTE*. 61–76.
[18] Simon S. Lam. 1988. Correction to "Protocol Conversion". *IEEE Trans. Software Eng.* 14, 9 (1988).
[19] Ivan Lanese, Fabrizio Montesi, and Gianluigi Zavattaro. 2015. The Evolution of Jolie. In *Essays Dedicated to M. Wirsing on the Occasion of His Retirement from the Chair of Programming and Soft. Eng. (Lect. Notes in Computer Science)*. 506–521.
[20] Fabrizio Montesi, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. 2007. JOLIE: a Java Orchestration Language Interpreter Engine. *Electronic Notes in Theoretical Computer Science* 181, Supplement C (2007), 19 – 33.
[21] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. 2002. Semantic Matching of Web Services Capabilities. In *The Semantic Web - ISWC 2002, First Int. Semantic Web Conf., Sardinia, Italy, Proceedings*. 333–347.
[22] 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). http://arxiv.org/abs/1407.0975
[23] Gwen Salaün, Tevfik Bultan, and Nima Roohi. 2012. Realizability of Choreographies Using Process Algebra Encodings. *IEEE Trans. Services Computing* 5, 3 (2012), 290–304. https://doi.org/10.1109/TSC.2011.9
[24] Bridget Spitznagel and David Garlan. 2003. A Compositional Formalization of Connector Wrappers. In *ICSE*.
[25] Roman Vaculín, Roman Neruda, and Katia P. Sycara. 2008. An Agent for Asymmetric Process Mediation in Open Environments.. In *SOCASE*.