On the Model-driven Synthesis of Evolvable Service 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

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 mismatching third-party services are to be composed, obtaining the distributed coordination and adaptation logic required to suitably realize a choreography is a non-trivial and error prone task. Automatic support is then needed. Nowadays, very few approaches address the problem of actually realizing choreographies in an automatic way. In this paper, we share the experience we had in two EU projects specifically targeted at choreographies, and report on a novel model-driven approach to the automatic synthesis of evolving choreographies we are currently working on. We illustrate our method at work on a use case in the domain of Smart Mobility & Tourism.

CCS CONCEPTS

• **Information systems → Web services**; • **Software and its engineering** → Model-driven software engineering; • **Computing methodologies → Distributed computing methodologies**;

KEYWORDS

Service choreographies, Coordination, Evolution

ACM Reference Format:
Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, Massimo Tivoli. 2018. On the Model-driven Synthesis of Evolvable Service Choreographies. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 [12, 19, 20, 24, 33] (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 [1–9, 14]. 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. In BPMN2 choreography diagrams, a participant role models the expected behavior (i.e., 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 that of *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 and adapt their interaction so to fulfill the collaboration prescribed by the choreography specification, in a fully distributed way.*

The approach consists of synthesizing coordination and adaptation software entities in order to proxify and control the interactions of the services participating to the choreography. When interposed among the services, the synthesized entities enforce the collaboration prescribed by the choreography specification. The ability to evolve the coordination logic in order to enable modular choreography evolution in response to possible context changes is a key factor. In this paper, we first describe the approach to the automatic synthesis of service choreographies we have developed within the CHOReVOLUTION project [1–3, 5–7, 9, 14]. Then, we focus on coordination aspects and describe the novel model-driven approach we are working on to manage choreography evolution through variability. The important body of knowledge on Software Product Lines (SPLs) [10, 16, 28, 38] is exploited for that purpose.

The paper is structured as follow. Section 2 sets the context and discusses variability aspects. Section 3 introduces an explanatory example. Section 4 presents the novel approach to manage choreography variability, 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

Software variability [28] is widely addressed in the SPLs [16] research domain. Within SPLs, at the level of the system's software architecture, variability is usually expressed by defining *variation points* [10, 38]. Each variation point represents a portion of the system that can be realized in different ways depending on, e.g.,

Conference'17, July 2017, Washington, DC, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

development decisions that can be taken only later in the development process, specific customizations of the system needed to satisfy different needs of different kinds of end users, in different contexts, or the availability of certain computing resources in the system's execution environment.

The specification of variation points is not natively supported by BPMN2 Choreography Diagrams, which lack of both explicit expressiveness of variability and ability to identify one or more points in the choreography model where a variation may occur [35]. A variation point may be needed when multiple choreography design options, i.e., *variants*, are available. According to this, for our purpose, a variation point specifies different choreography variants that, during the runtime, can be active or not depending on the context. In order to make the paper self-contained, in the following we briefly describe the process we defined in the CHOReVOLUTION project for the synthesis of choreographies. Then, in Section 4, we describe how the synthesis process is being updated to support choreography evolution through variation points.

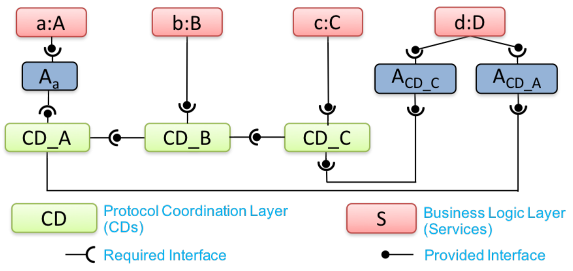

Figure 1: Architectural style (a sample instance of)

The synthesis process consists of a set of core *code generation phases*. It takes as input a choreography specification together with a set of services as possible candidates to play the choreography roles, and automatically generates a set of coordination software entities. When interposed among the services according to a predefined architectural style (see Figure 1), these software entities "proxify" the participant services to coordinate their interaction, when needed. Specifically, coordination entities are called *Coordination Delegates* (CDs) and guarantee the collaboration specified by the choreography specification through distributed protocol coordination.

The coordination logic of the CDs is concrete service independent, meaning that it is synthesized by considering the expected interaction protocol specified for the roles instead of the actual 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. Then, service *Adapters* are synthesized in order to correctly bind concrete services to abstract roles. Specifically, adapters solve possible protocol mismatches between services and choreography roles using Enterprise Integration Patterns [22] (EIP) as adaptation primitives, and composing them to realize the required adaptation logic. More details on the synthesis process can be found in [6] (and references there in). The synthesis processor is fully implemented as part of the CHOReVOLUTION Studio[3] [4].

The novel contribution of this paper concerns both (i) the extension of the metamodel underlying the BPMN2 Modeler[5] that we use for specifying choreography diagrams, and (ii) the definition of a new version of coordination delegates, namely, *evolvable CDs* (eCDs). At design time, the extended BPMN2 Modeler will allow to specify variation points directly within choreography diagrams. At runtime, eCDs will permit to handle choreography evolution in response to context changes.

3 EXPLANATORY EXAMPLE

This section introduces an explanatory example in the domain of Smart Mobility & Tourism (SMT). The related BPMN2 choreography diagram is shown in Figure 2.

The main element of a choreography diagram is the choreography task (e.g., `Get Tourist Guide` task on top of Figure 2). 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, i.e., the one deciding 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 System (CTAS) through the cooperation of several content and service providers, organizations and authorities. It envisages a mobile application as an "Electronic Touristic Guide" that interacts with the system 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 initiaties 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 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 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 THE APPROACH

The challenges regarding the specification and the handling of variability for choreographies have also been studied in [35]. For what concerns CHOReVOLUTION, the challenges we have addressed can

[3]https://projects.ow2.org/view/chorevolution/
[4]https://gitlab.ow2.org/chorevolution
[5]https://www.eclipse.org/bpmn2-modeler/

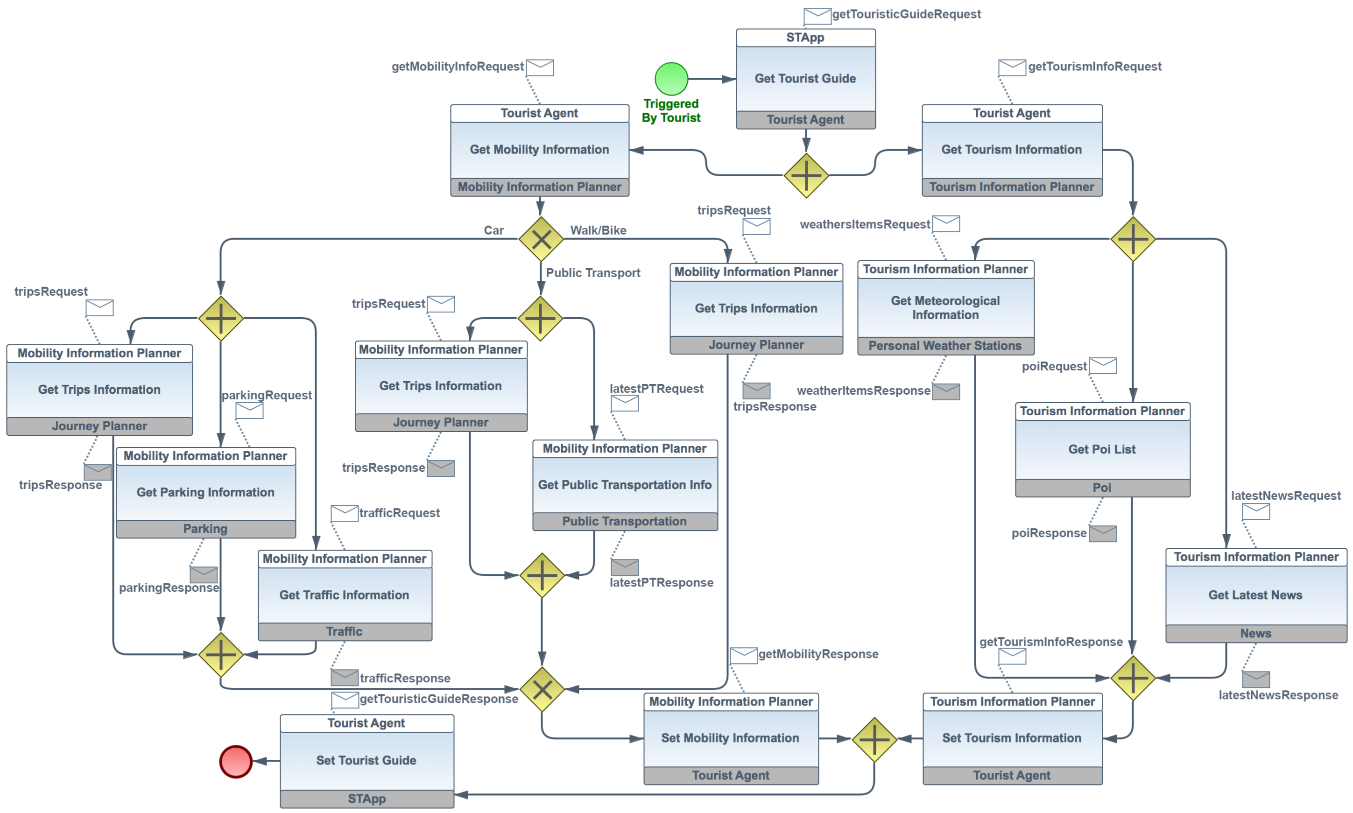

Figure 2: SMT choreography specification

be summarized as follows: **[C1.]** lack of constructs for explicitly specifying variation points and variants in BPMN2 choreography diagrams; **[C2.]** lack of support for specifying associations of variation points with variants and related contexts; **[C3.]** lack of automatic support for generating the code in charge of managing the different variants without disrupting the choreography execution.

Figure 3 shows the portion of BPMN2 metamodel that we have extended in order to support the specification of choreography variation points and variants. In line with the approach in [23], choreography variation points can be specified only for one type of BPMN2 choreography diagram construct. That is, we defined a new metaclass named ChoreographyVariationPoint as an extension of the metaclass ChoreographyActivity. The metaclass ChoreographyVariant, which in turn extends the metaclass Choreography, describes a possible choreography variant. The definition of these metaclasses solves the challenge C1. The composition relationship between ChoreographyVariationPoint and ChoreographyVariant allows the choreography modeler to define from 1 to n variations associated to a variation point. Thus, the challenge C2 is also solved. The association between the metaclass ChoreographyVariationPoint and the BPMN2 metaclass Participant permits to specify the participants that initiate the choreography variants. By following the BPMN2 basic validation rule for correctly sequencing choreography activities [29], for a variation point, the initiator participants must have been involved

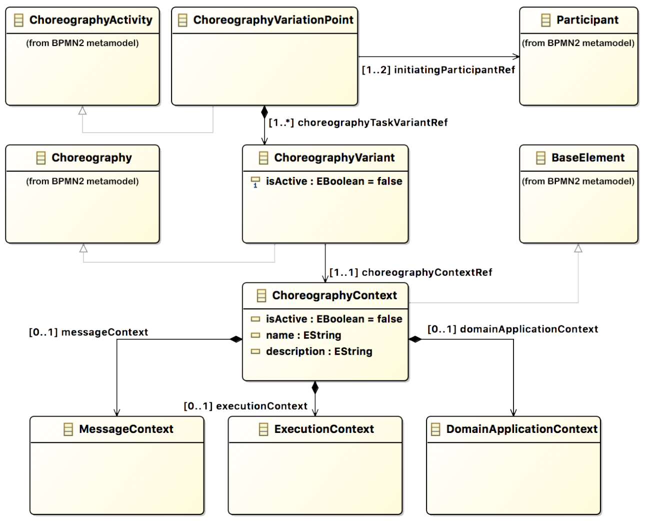

Figure 3: Variation point metamodel

(as initiators or receivers) in the choreography activities preceding the variation point.

The source of the choreography evolution is the dynamically "sensed" choreography context. Concretely, at run-time, context changes trigger the activation/deactivation of variation points and related variants by checking aptly defined context conditions. As it will be clear soon, only one variation can be enabled at run-time and the execution of a variation is treated as atomic by CDs. Based on this, the bottommost side of the metamodel defines the choreography context as an extension of `BaseElement` metaclass by distinguishing among:

- `MessageContext` – it is described in terms of data contained in the messages exchanged among the involved services up to a given point in time.
- `ExecutionContext` – it is described by using information about possible computational resources. Thus, in order to sense an execution context, dedicated functionalities are provided by the run-time support offered by the CHOReVOLUTION middleware and cloud infrastructure layers.
- `DomainApplicationContext` – it is characterized by conditions on the surrounding environment, given that dedicated functionalities (e.g., supported by dedicated sensors) are provided to sense it.

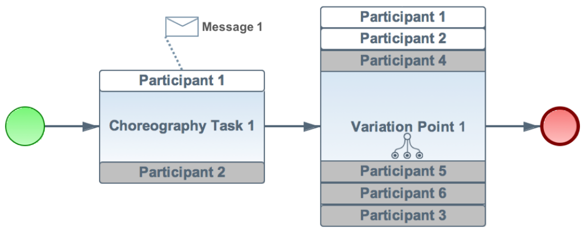

Figure 4: Variation point

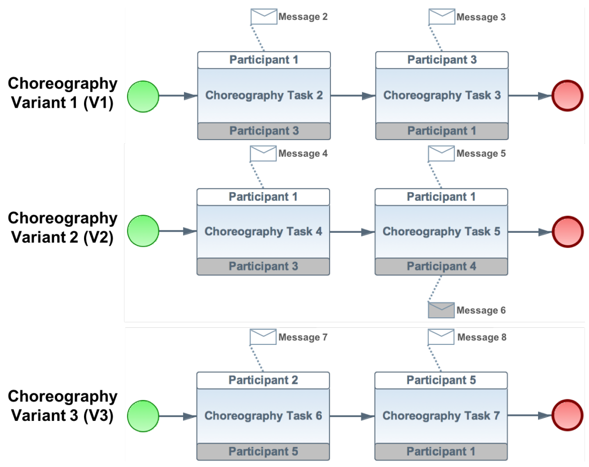

Figure 5: Choreography variants

Figure 4 shows a sample choreography consisting of one task (`Choreography Task 1`), and one variation point (`Variation Point 1`). The latter specifies three possible choreography variants, namely `V1`, `V2`, and `V3` (see Figure 5). The initiating participants of the variation point are both Participant 1 and 2 (see the white bands), since Participant 1 initiates the interaction of the variants V1, V2, and Participant 2 initiates the interaction of the variant V3. The choreography variant `V1` specifies two tasks: `Choreography Task 2` involving Participant 1 and Participant 3 exchanging the message `Message 2`; `Choreography Task 3` involving Participant 3 and 1 exchanging the message `Message 3`. Note that, in order to enable the specification of variation points, new graphical elements are introduced in the CHOReVOLUTION Studio by using the BPMN2 Modeler extension API [30] (see the marker under the task label).

The ability to specify variations points permits to deal with a form of choreography evolution through an enhanced notion of CDs, called *evolvable CDs* (eCDs). Beyond performing pure coordination logic, eCDs are able to manage choreography variation points by activating/deactivating the related choreography variants according to the sensed choreography context. In order to guarantee the correct execution of the choreography activities related to a variation point, the coordination logic of the CD associated to the initiating participant of the choreography task preceding the choreography variation point is extended so to be able to interact with the eCD managing the variation point. The ability to automatically synthesize the code of eCDs solves the challenge C3.

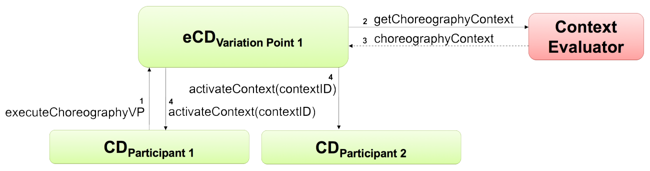

Figure 6: CD interactions on variation points

Figure 6 exemplifies the CDs interaction pattern corresponding to the choreography variation point in Figure 4, and the related variations in Figure 5. The scenario is as follows: CD$_{Participant\ 1}$ and CD$_{Participant\ 2}$ have just coordinated the exchange of `Message 1` so to realize `Choreography Task 1`, i.e., the task immediately preceding the variation point. Then, CD$_{Participant\ 1}$ interacts with eCD$_{Variation\ Point\ 1}$ (step 1), which in turn interacts with the *Context Evaluator* in order to evaluate the context (steps 2 and 3). According to the response of the *Context Evaluator*, eCD$_{Variation\ Point\ 1}$ notifies the CD that is allowed to initiate the execution of the context-enabled variant. That is, eCD$_{Variation\ Point\ 1}$ notifies either CD$_{Participant\ 1}$ or CD$_{Participant\ 2}$ by sending a message containing the choreography context identifier (step 4). It is worthwhile to recall that the context conditions are exclusive, meaning that only one variation at a time can be enabled.

5 VARIATION POINTS AT WORK

In order to demonstrate the approach at work, Figures 7 shows a modified version of the SMT choreography containing a variation point and two variants. As already said in Section 3, one of the parallel execution flows of the SMT choreography is responsible for retrieving touristic information. Part of this information concerns the points of interest (POI) to be visited by a tourist. The variation point `Retrieve Tourism Information` specifies two variants that are activated/deactivated based on the weather forecast.

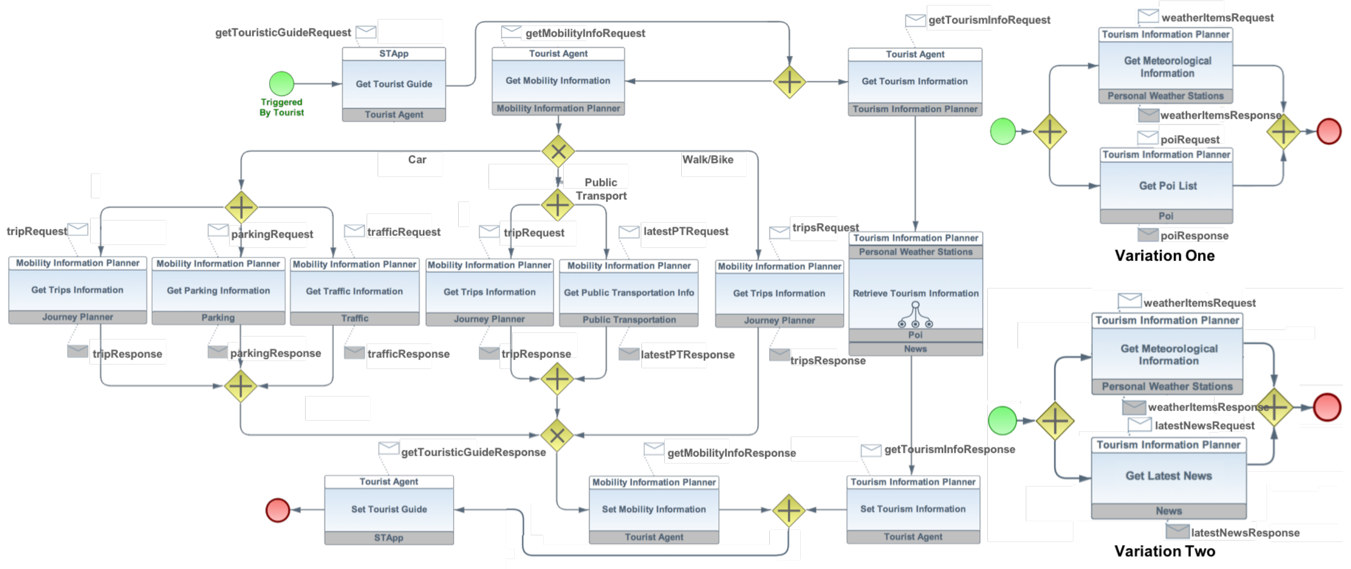

Figure 7: SMT choreography with a variation points and two variants

The choreography corresponding to Variant One involves the interactions among Tourism Information Planner, Personal Weather Stations, and Poi. This choreography variant prescribes the retrieval of outdoor POIs and, according to the evaluation of the associated domain context, it is activated when it is not raining.

Variant Two describes a choreography involving Tourism Information Planner, Personal Weather Stations, and News. This variant concerns the retrieval of a list of indoor events (e.g., art exhibitions, visits to museums), and it is activated when it is raining.

As shown in Figure 8, upon reaching the choreography variation point `Retrieve Tourism Information`, $CD_{Tourism\ Information\ Planner}$ interacts with the evolvable CD $eCD_{Retrieve\ Tourism\ Information}$ by calling the operation `executeChoreographyVP`. Then, after retrieving the choreography context, $eCD_{Retrieve\ Tourism\ Information}$ notifies the choreography context to $CD_{Tourism\ Information\ Plannner}$ and the execution of the choreography variant corresponding to the retrieved context is enabled.

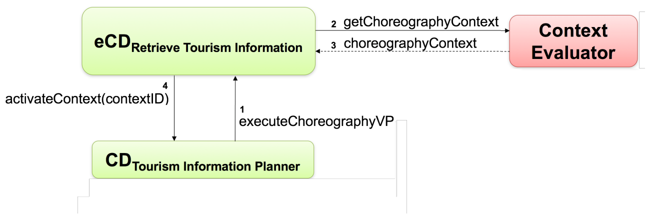
eCDRetrieve Tourism Information
2 getChoreographyContext
3 choreographyContext
Context Evaluator
4 activateContext(contextID)
1 executeChoreographyVP
CDTourism Information Planner

Figure 8: Retrieve tourism information variation point

Figure 9 shows the resulting choreography architecture automatically generated by the synthesis process. Notably, $CD_{Tourism\ Information\ Plannner}$ is linked with the newly added $eCD_{Retrieve\ Tourism\ Information}$, which in turn is connected to the Context Evaluator.

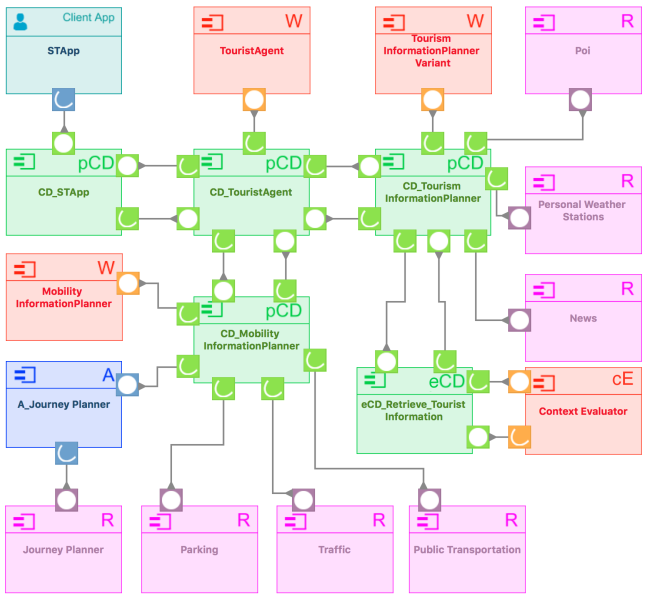
Client App
STApp
W
TouristAgent
Tourism InformationPlanner Variant
R
Poi
pCD
CD_STApp
CD_TouristAgent
CD_Tourism InformationPlanner
Personal Weather Stations
Mobility InformationPlanner
CD_Mobility InformationPlanner
News
A
A_Journey Planner
eCD
eCD_Retrieve_Tourist Information
cE
Context Evaluator
Journey Planner
Parking
Traffic
Public Transportation

Figure 9: Architecture configuration with evolvable CDs

6 RELATED WORK

Software Product Lines (SPL) offers a systematic way for managing variability. Several approaches have been introduced to facilitate the development of variable service architectures in the context of service-oriented SPLs [13, 17, 18, 26, 27, 31, 32, 36, 37].

Several variability models are proposed to capture, organize and represent variability. A review of variability model categorizations can be found in [15]. For example, COVAMOF-like [34], OVM-like [11], and CVL-like [21] variation models can be used to represent service variability. However, these models do not provide a mechanism to consistently map choreography variability specifications to actual code.

Most closely related to our work, in [35], the authors propose XChor, a choreography language for integrating variable orchestration specifications. Differently from us, the authors do not propose an extension dedicated to BPMN2 choreography diagrams that, nowadays, represent the de-facto standard for specifying choreographies. In [35], the authors also study state-of-the-art orchestration and choreography languages. The study concludes that existing languages do not address interface and composition variability explicitly. Only the VxBPEL [25] language permits explicit variability specifications based on the COVAMOF [34] model. It enables the definition of variability in service composition, variability in interface and connector is not permitted.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have reported on a novel model-driven approach to the automatic synthesis of evolving choreographies we are currently working on. We have described our method at work on an explanatory example in the Smart Mobility & Tourism domain. The application to this example shows that the approach is viable towards proposing a practical approach to choreography evolution.

The ongoing implementation of the described method is part of a model-based tool chain released as an open-source project[6], and it is supported by the OW2 Future Internet Software and Services initiative (FISSi[7]).

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, Paola Inverardi, Filippo Mignosi, Romina Spalazzese, and Massimo Tivoli. 2015. Automated Synthesis of Application-Layer Connectors from Automata-Based Specifications. In *LATA 2015*.

[2] Marco Autili, Paola Inverardi, and Massimo Tivoli. 2014. CHOREOS: Large scale choreographies for the future internet. In *CSMR-WCRE 2014*.

[3] Marco Autili, Paola Inverardi, and Massimo Tivoli. 2015. Automated Synthesis of Service Choreographies. *IEEE Software* (2015), 50–57.

[4] 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* (2008), 2210–2236.

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

[6] Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, and Massimo Tivoli. 2018. Model-driven adaptation of service choreographies. In *SAC 2018*.

[7] 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 *FOCLASA 2015*.

[8] Marco Autili, Amleto Di Salle, and Massimo Tivoli. 2013. Synthesis of Resilient Choreographies. In *SERENE 2013*.

[9] Marco Autili and Massimo Tivoli. 2014. Distributed Enforcement of Service Choreographies. In *FOCLASA 2014*.

[10] Muhammad Ali Babar, Lianping Chen, and Forrest Shull. 2010. Managing Variability in Software Product Lines. *IEEE Software* (2010), 89–91.

[11] Felix Bachmann, Michael Goedicke, Julio Cesar Sampaio do Prado Leite, Robert L. Nord, Klaus Pohl, Balasubramaniam Ramesh, and Alexander Vilbig. 2003. A Meta-model for Representing Variability in Product Family Development. In *PFE 2003*.

[12] Samik Basu and Tevfik Bultan. 2011. Choreography conformance via synchronizability. In *WWW 2011*.

[13] Nicola Boffoli, Marta Cimitile, Fabrizio Maria Maggi, and Giuseppe Visaggio. 2009. Managing SOA System Variation through Business Process Lines and Process Oriented Development.

[14] Radu Calinescu, Marco Autili, Javier Cámara, Antinisca Di Marco, Simos Gerasimou, Paola Inverardi, Alexander Perucci, Nils Jansen, Joost-Pieter Katoen, Marta Z. Kwiatkowska, Ole J. Mengshoel, Romina Spalazzese, and Massimo Tivoli. 2017. Synthesis and Verification of Self-aware Computing Systems. In *Self-Aware Computing Systems*.

[15] Lianping Chen, Muhammad Ali Babar, and Nour Ali. 2009. Variability management in software product lines: a systematic review. In *SPLC 2009*.

[16] Patrick Donohoe. 2012. Introduction to software product lines. In *SPLC 2012*.

[17] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan. 2009. Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments *(LNCS)*, Vol. 5525. Springer, 146–163.

[18] Kurt Geihs, Roland Reichle, Michael Wagner, and Mohammad Ullah Khan. 2009. Service-Oriented Adaptation in Ubiquitous Computing Environments. In *CSE 2009*.

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

[20] Sylvain Hallé and Tevfik Bultan. 2010. Realizability analysis for message-based interactions using shared-state projections. In *ACM SIGSOFT 2010*.

[21] Øystein Haugen, Andrzej Wasowski, and Krzysztof Czarnecki. 2013. CVL: common variability language. In *SPLC 2013*.

[22] 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.

[23] Paul Istoan, Grégory Nain, Gilles Perrouin, and Jean-Marc Jézéquel. 2009. Dynamic Software Product Lines for Service-Based Systems. In *CIT 2009*.

[24] Raman Kazhamiakin and Marco Pistore. 2006. Analysis of Realizability Conditions for Web Service Choreographies. In *IFIP WG 2006*.

[25] Michiel Koning, Chang-ai Sun, Marco Sinnema, and Paris Avgeriou. 2009. VxBPEL: Supporting variability for Web services in BPEL. *Information & Software Technology* (2009), 258–269.

[26] Ivan Lanese, Antonio Bucchiarone, and Fabrizio Montesi. 2010. A Framework for Rule-Based Dynamic Adaptation. In *TGC 2010*.

[27] Bardia Mohabbati, Marek Hatala, Dragan Gasevic, Mohsen Asadi, and Marko Boskovic. 2011. Development and configuration of service-oriented systems families. In *(SAC) 2011*.

[28] Tuan Nguyen, Alan W. Colman, Muhammad Adeel Talib, and Jun Han. 2011. Managing service variability: state of the art and open issues. In *VAMOS 2011*.

[29] Object Management Group. 2018. Documents Associated With Business Process Model And Notation. http://www.omg.org/spec/BPMN/2.0.2/PDF.

[30] Eclipse Project. [n. d.]. BPMN2-Modeler Model Extension. https://wiki.eclipse.org/BPMN2-Modeler/DeveloperTutorials/ModelExtension, year=2018.

[31] Maryam Razavian and Ramtin Khosravi. 2008. Modeling variability in the component and connector view of architecture using UML. In *AICCSA 2008*.

[32] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein O. Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. 2009. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments *(LNCS)*, Vol. 5525. Springer, 164–182.

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

[34] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. 2004. COVAMOF: A Framework for Modeling Variability in Software Product Families. In *SPLC 2004*.

[35] Selma Suloglu, Bedir Tekinerdogan, and Ali Dogru. 2013. XChor Choreography Language for Integration of Variable Orchestration Specifications. In *BMSD 2013*.

[36] Hongyu Sun, Robyn R. Lutz, and Samik Basu. 2009. Product-line-based requirements customization for web service compositions. In *SPLC 2009*.

[37] N. Yasemin Topaloglu and Rafael Capilla. 2004. Modeling the Variability of Web Services from a Pattern Point of View. In *ECOWS 2004*.

[38] Diana L. Webber and Hassan Gomaa. 2004. Modeling variability in software product lines with the variation point model. *Sci. Comput. Program.* (2004).

[6] https://projects.ow2.org/view/chorevolution/
[7] https://www.ow2.org/view/Future_Internet