Towards Adapting Choreography-Based Service Compositions Through Enterprise Integration Patterns

Amleto Di Salle, Francesco Gallo, and Alexander Perucci(✉)

University of L'Aquila, Via Vetoio, 67100 L'Aquila, Italy
{amleto.disalle,francesco.gallo}@univaq.it,
alexander.perucci@graduate.univaq.it
http://www.univaq.it

Abstract. The Future Internet is becoming a reality, providing a large-scale computing environments where a virtually infinite number of available services can be composed so as to fit users' needs. Modern service-oriented applications will be more and more often built by reusing and assembling distributed services. A key enabler for this vision is then the ability to automatically compose and dynamically coordinate software services. Service choreographies are an emergent Service Engineering (SE) approach to compose together and coordinate services in a distributed way. 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. In this direction, this paper leverages previous work on the automatic synthesis of choreography-based systems, and describes our preliminary steps towards exploiting Enterprise Integration Patterns to deal with a form of choreography adaptation.

1 Introduction

The Future Internet promotes a distributed computing environment that will be increasingly surrounded by a large number of software services, which can be composed to meet user needs. The Future Internet of Services paradigm emerges from the convergence of the Future Internet (FI) and the Service-Oriented Computing (SOC) paradigm [1]. Services play a central role in this vision as effective means to achieve interoperability between heterogeneous parties of a business process, and new value added service-based systems can be built as a *choreography* of services available in the FI. Service choreography is a decentralized approach, which provides a loose way to design service composition by specifying the participants (i.e., roles) and the (message-based) interaction protocol between them, by decoupling the participant tasks from the services that only later will be bound to the specified roles.

The need for service choreography was recognized in the Business Process Modeling Notation version 2.0[1] (BPMN2), which introduced *Choreography*

[1] http://www.omg.org/spec/BPMN/2.0/.

DOI: 10.1007/978-3-662-49224-6_20

Diagrams to offer choreography-specific modeling constructs. A choreography diagram models peer-to-peer communication by defining a multi-party protocol that, when put in place by the cooperating parties, will permit to reach the overall choreography objectives in a fully distributed way. In this sense, service choreographies are quite different from service orchestrations in which a single stakeholder centrally plans and decides how an objective should be reached through the cooperation with other services.

In this paper we leverage the experience on choreography development that we have been doing so far within the EU CHOReOS project[2]. Then, being supported by the EU CHOReVOLUTION (follow-up) project[3], we report on the novel idea we are currently investigating to achieve choreography adaptation and evolution to face the challenges posed by the heterogeneity of FI services.

In this direction, we propose a way to enhance the previous CHOReOS approach to the automatic synthesis of choreography-based systems [2–5], and describes the preliminary steps we are undertaking within CHOReVOLUTION towards exploiting Enterprise Integration Patterns (EIP) so as to deal with a form of choreography adaptation. The novel contributions can be summarized as follow: (i) adoption of EIP to deal with a form of adaptation for choreography-based systems; (ii) enhancement of our synthesis process by introducing an adapters generator; (iii) enhancement of the architectural style for including adapters.

The paper is structured as follow. Section 2 sets the context of our work, and Sect. 3 introduces an explanatory example. Then, Sect. 4 describes how the synthesis process can be enhanced to deal with choreography adaptation and evolution through protocol coordination, protocol adaptation and related complex data mappings, and Sect. 5 describes the proposed enhancement at work on the explanatory example. Related work is discussed in Sect. 6, and conclusions are given in Sect. 7.

2 Setting the Context

This section sets the context of our work by describing the problem we want to address in Sect. 2.1, and the idea underlying the proposed solution in Sect. 2.2. Then, Sect. 2.3 provides basic notions of the Enterprise Integration Patterns (EIP) [6] that we propose to exploit to deal with adaptation issues.

2.1 The Problem Space

When considering choreography-based service-oriented systems, the following problems are mainly considered:

(i) *realizability check* - checks whether the choreography can be realized by implementing each participant so that it conforms to the played role;

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

(ii) *conformance check* - checks whether the set of services satisfies the choreography specification;
(iii) *automatic realizability enforcement* - given a choreography specification and a set of existing services, externally coordinate and adapt their interaction so as to fulfill the collaboration prescribed by the choreography specification.

In the literature, the approaches proposed in [7–19] address the problems (i) and (ii); the approaches proposed in [2,3,5,20] address the problem (iii). In this paper we concentrate on the automatic realizability enforcement problem. Specifically, starting from previous work in [2–5], we propose the following enhancement to deal with a form of choreography adaptation that exploits EIP to built service adapters.

2.2 The Solution Space

Addressing the automatic realizability enforcement problem calls for solving both coordination issues and adaptation issues.

Coordination issues are addressed in previous work [2,5], where we propose an automatic approach to synthesize the global coordination logic to be then distributed and enforced among the considered services. Preliminary ideas towards addressing adaptation issues are described in [3,4], where we propose the use of adapters for solving interaction protocol mismatches deriving from the heterogeneity of services not born to be directly composed together.

In this paper we describe the initial steps we have done towards exploiting Enterprise Integration Patterns so as to deal with a form of choreography adaptation that, in addition to interaction protocol mismatches, also account for I/O data mismatches. Our mid-term goal within the CHOReVOLUTION project is to achieve automated data-flow coordination and adaptation, which means effectively coping with heterogeneous service interfaces and dealing with as much EIPs [6] as possible in a automatic way. In particular, the idea is to automatically generate adapters by combining different EIPs based on a notion of protocol mediation and data similarity.

2.3 Exploiting Enterprise Integration Patterns

From a technical point of view, achieving the above calls for dealing with *mismatching service signatures* and *interaction protocols*. In particular, to achieve adaptation, the operations signature and the interaction protocol of the concrete services may need to be adapted to the roles to be played in the input choreography model. This requires to implement a suitable notion of matching between protocols by means of *complex data mappings* over both operation names and I/O messages. Protocol refinement techniques must be developed to bridge the gap between the abstract protocol of the choreography participant roles and the protocol of the concrete services. These techniques, together with the ability of dealing with, e.g., appearing and disappearing services at run-time, would permit to achieve evolution through on-the-fly service binding.

EIPs offer more than one approach for integrating applications, i.e., *File Transfer*, *Shared Database*, *Remote Procedure Invocation*, and *Messaging* [6]. We focus on the Messaging approach since we consider Web Services (WSs) as possible choreography participants, and WSs communicate through messages passing (e.g., request/response or one-way operation types).

The Messaging approach uses the "pipes-and-filters" architectural style [21] as base for connecting applications. The Endpoints (Filters) are connected with one another via Channels (Pipes). The producing endpoint sends messages to the channel, and the messages are retrieved by the consuming endpoint. There are different types of pipes and filters patterns, each one of them dedicated to solve a particular integration aspect.

For the purposes of this paper, we consider: *Message Transformation* that converts a message from a format to another one; *Message Aggregator* that receives multiple messages and combines them into a single message.

3 Explanatory Example

The explanatory example introduced in this section is a very small portion of an *In-store Marketing and Sale* choreography that was used by the EU CHOReOS project to demonstrate an *Adaptive Customer Relationship Booster* system. The whole choreography was aimed at monitoring the activity of a client inside the shop in order to propose him/her tailored shopping offers and/or advertisements according to the user information (preferences, current shopping list, etc.) held by a shopping assistant application service.

Figure 1 reports a simplified choreography diagram realized by using the Eclipse BPMN2 modeler plugin[4]. The diagrams also shows the input and output messages of each choreography task. Within the Eclipse BPMN2 modeler, messages are specified by using the XML schema, which is the default language for specifying BPMN2 messages.

The choreography is triggered by the `Client` entering the shop. A `Shop Entrance` service (not shown in the figure) detects the presence of a specific `Client` inside the store and assigns him a virtual cart. Once subscribed to the cart, the `Client` can add and remove products to and from it. Once the `Client` finishes shopping, the `Smart Cart` service allows for executing the payment by interacting with the a `Self Check-out Machine`.

4 Method Description

In this section we describe the proposed method by distinguishing between *protocol coordination* and *protocol adaptation*.

[4] http://www.eclipse.org/bpmn2-modeler/.

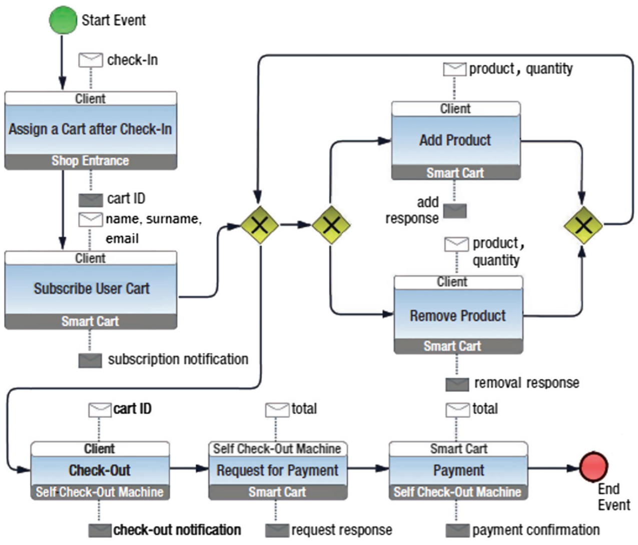

Fig. 1. In-store marketing and sale choreography

Protocol coordination allows for preventing undesired interactions among (possibly adapted) services. That is, interactions not allowed by the choreography specification can happen when the services collaborate in an uncontrolled way. To deal with this problem, additional software entities, called *Coordination Delegates* (CDs), are generated and interposed among the services participating in the specified choreography in order to prevent possible undesired interactions. Thus, the intent of CDs is to coordinate the interaction of the participant services in a way that the resulting collaboration correctly realizes the specified choreography. For instance, the `Client` is allowed to perform the `Add Product` task to add products to the `Smart Cart` (see the top of the Fig. 1). However, after paying and before check-out, an undesired interaction can happen since the `Client` might try to add products (see the top-most tasks just before the End Event), thus avoiding paying for them.

Protocol adaptation allows for dealing with services that do not exactly fit the choreography roles. That is, adapters are automatically synthesized to mediate the interaction service-to-CD and CD-to-service according to the choreography roles (see Fig. 2). Each Adapter is generated so as to bridge/mediate the concrete service interaction protocol in order to exactly match the abstract participant interaction protocol. In other words, Adapters realize correct service-role binding by solving possible interoperability issues (e.g., signature and protocol mismatches) between concrete services and abstract participants. By leveraging a sufficiently accurate notion of behavioral interface refinement, Adapters enforce service-role similarity, hence binding the concrete services to the abstract roles defined by the choreography. The synthesized Adapters enforce exact similarity through complex data mappings and complex protocol mediation patterns. For instance, Adapters are able to map message data types, or reorder/merge/split the sequence of operation calls and/or related I/O messages.

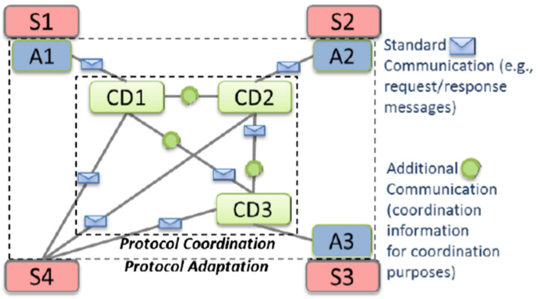
S1
S2
A1
A2
CD1
CD2
CD3
A3
S4
S3
Protocol Coordination
Protocol Adaptation
Standard Communication (e.g., request/response messages)
Additional Communication (coordination information for coordination purposes)

Fig. 2. Architectural style with adapters

Coordination and adaptation software entities are synthesized in order to proxify and control the participant services' interaction. When interposed among the services, according to the architectural style shown in Fig. 2, *coordination entities* still guarantee the collaboration specified by the choreography specification through protocol coordination; *adaptation entities* mediate the interaction of the participant services so as to fit the choreography roles.

An important aspect here is that the coordination logic performed by the CDs is *service-independent* since it is based on the expected behavior of the participants as specified by the choreography, rather than on the actual concrete services to be binded and coordinated. In this way *separation of concerns* is realized by separating pure coordination issues (i.e., undesired interactions) from adaptation/mediation ones (e.g., operation signature mismatches and data incompatibilities at the service interface level, and behavior mismatches). For example, the latter can arise whenever a service discovered as a participant does not exactly match the role to be played.

In order to automatically synthesize adaptation software entities we propose an extension of our `CHOReOSynt` tool [22] introducing a new RESTful service called `Synthesis Adapter Generator` (see Fig. 3).

By taking as input a BPMN 2.0 specification of the choreography, the extension we propose allows for deriving service Adapters in addition to CDs (Fig. 3). To this end, model transformations are employed and interoperation with the `Service Discovery` is required (out of the scope of this paper). Both CDs and Adapters, when deployed by the `Enactment Engine` (out of the scope of this paper), allow for enacting the choreography by realizing the distributed coordination logic between the discovered services.

The tool consists of the following RESTful services, and a set of Eclipse plugins that have been developed to interact with such services.

M2M Transformator – The Model-to-Model (M2M) Transformator offers a set of model transformations.

Synthesis Discovery Manager – The Synthesis process and the Discovery process interact each other to retrieve, from the service base, those candidate services that are suitable for playing the participant roles required by the chore-

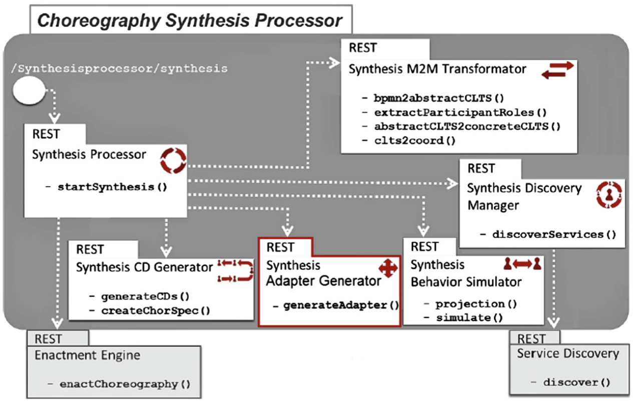

Fig. 3. REST architecture of the extended synthesis processor

ography specification, and hence, those services whose (offered and required) operations and behavior are compatible with the expected behavior as extracted from the choreography through projection.

Behavior Simulator – Once a set of concrete candidate services has been discovered, the synthesis process has to select them by checking, for each participant, if its expected behavior can be simulated by some candidate service. Note that, for a given participant, behavioral simulation is required since, although the discovered candidate services for it are able to offer and require (at least) the operations needed to play the role of the participant, one cannot be sure that the candidate services are able to support the operations flow as expected by the choreography.

Coordination Delegate and Adapter Generators – Once the services have been selected for all the choreography participants, the synthesis processor can generate the needed CDs and Adapters through the operations `generateCD()` and `generateAdapter()`, respectively.

In the following we introduce an example in the marketing and sale domain that will be then used in Sect. 5 to describe our method at work.

5 Method at Work

This section describes the proposed enhancement at work on the explanatory example introduced in Sect. 3. There are several frameworks and/or systems that implement/use EIPs in order to integrate applications. We have chosen Spring Integration framework[5] since it implements most of the EIPs, and it is well integrated with the Spring ecosystem. In particular, it is integrated with the Spring Web Services project[6].

[5] http://projects.spring.io/spring-integration/.
[6] http://projects.spring.io/spring-ws/.

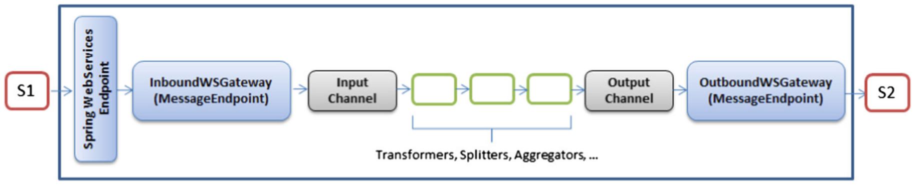
S1
Spring WebServices Endpoint
InboundWSGateway (MessageEndpoint)
Input Channel
Output Channel
OutboundWSGateway (MessageEndpoint)
S2
Transformers, Splitters, Aggregators, ...

Fig. 4. Adapter architecture

Figure 4 describes the architecture of the generated adapters by using Spring Web Services and Spring Integration. In particular, the Spring Web Services Endpoint is the Web Service that mediates the interaction of the Service `S1` and the Service `S2`. When the Service `S1` calls an operation *op1* by sending a message *m1*, the Endpoint receives the operation and put the message into the input channel by using Inbound Web Service Gateways. The chain of EIPs, from the Input Channel to the Output Channel, is generated by the synthesis processor depending of the found interoperability issues (e.g., signature and protocol mismatches). The chain is made of one or more EIPs handlers to, e.g., *Message Transformers*, used to convert a message from one format to another one; *Message Routers*, used to decouple a message source from the ultimate destination of the message, and so on. *Message Routers* patterns can be, e.g., *Splitter*, *Aggregator*, *Resequencer* [6].

Referring to the explanatory example in Fig. 1, we focus on the `Subscribe User Cart` and `Add Product` Choreography Tasks.

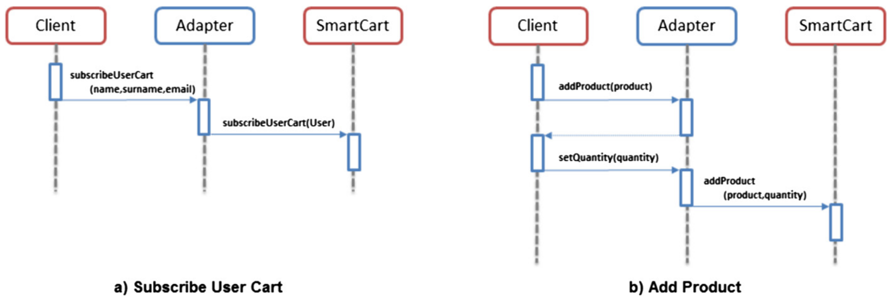

Fig. 5. Adapter example

Concerning `Subscribe User Cart` choreography task, let us suppose that the `Client` service is able to invoke a `subscribeUserCart` operation expecting as input message three string elements, one for `name`, one for `surname`, and one for `email`. The XSD schema codifying the input message is shown in Listing 1.1. Let us also suppose that the `SmartCart` service offers a `subscribeUserCart` operation expecting as input message only one `User` element. As shown in Listing 1.2, this element is a complex type encapsulating the following string elements: `firstname`, `lastname`, and `mail`.

In order to let the `Client` and the `SmartCart` services to communicate, the processor generates an adapter that offers the operation `subscribeUserCart (name, surname, email)` so that when the `Client` invokes `subscribeUserCart (name, surname, email)` operation, the adapter transforms the first message (Listing 1.1) into the second one (Listing 1.2). This is done by generating an ad-hoc *Message Transformer* handler and adding it to the chain. At the end, the adapter invokes the `subscribeUserCart(User)` operation offered by the `Smart Cart` service. This behavior is shown in Fig. 5a.

Listing 1.1. input parameters of subscribeUserCart operation

```
<xsd:schema version="1.0" targetNamespace="http://choreosynth.disim.univaq.it/">
  <xsd:element name="name" type="xsd:string"></xsd:element>
  <xsd:element name="surname" type="xsd:string"></xsd:element>
  <xsd:element name="email" type="xsd:string"></xsd:element>
</xsd:schema>
```

Listing 1.2. input parameters of subscribeUserCart operation discovered

```
<xsd:schema version="1.0" targetNamespace="http://choreosynth.disim.univaq.it/">
  <xsd:complexType name="User">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string"></xsd:element>
      <xsd:element name="lastname" type="xsd:string"></xsd:element>
      <xsd:element name="mail" type="xsd:string" minOccurs="0" maxOccurs="unbounded">
          ↪</xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Concerning the `Add Product` choreography task let us suppose that the `Client` service invokes two operations, `addProduct(product)` and `setQuantity (quantity)`. Let us also suppose that the `SmartCart` service offers a `addProduct (product,quantity)` operation. Differently from the previous case, the adapter is now generated by using the *Message Router Aggregator* pattern. This pattern allows for accumulating the two messages (i.e., `product` and `quantity`) received from the `Client`, and subsequently invokes the `addProduct (product, quantity)` operation offered by `SmartCart` (as shown in the Fig. 5b).

The method for generating adapters exemplified above requires automated synthesis of I/O data mappings. To this end, the idea is to exploits a slightly modified version of the Strawberry tool [23] that allows for automatically inferring data mappings between different messages of two different Web services, i.e., `Client` and `SmartCart` in our case. Strawberry exploits (i) static data type analysis to analyze the type structure[7] of the two different messages; (ii) testing check if the two messages are also semantically correlated (since in general, considering the messages' type structure only is not sufficient). Efforts in this direction will be part of future work.

[7] E.g., the type structure of the XML Schema types of the messages in the WSDL of the considered services.

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 [24,25].

Recently, with the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [26,27]. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

Spitznagel and Garlan present an approach for formally specifying adapter wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches [28]. Although this formalizations supports modularization, automated synthesis is not treated at all hence keeping the focus only on adapter design and specification.

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 [29]. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adapter, hence delegating to the user a non-trivial specification task.

Recently, Bennaceur and Issarny presented an approach that, exploiting ontology reasoning and constraint programming, allows for automatically inferring mappings between components interfaces [30]. Importantly, these mappings guarantee semantic compatibility between the operations and data.

Rahm et al. propose a catalog of criteria for documenting the evaluations of schema matching systems [31]. In particular, the authors discuss various aspects that contribute to the match quality obtained as the result of an evaluation. In [32,33] 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 [34] 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.

7 Conclusion and Future Works

In this paper, we propose a way to enhance the previous CHOReOS approach to the automatic synthesis of choreography-based systems, and we report on the novel idea we are currently investigating within CHOReVOLUTION to achieve choreography adaptation and evolution. In particular, the idea is to automatically generate adapters by combining different EIPs depending on a notion of protocol mediation and data similarity. In order to automatically synthesize the adapters we propose an extension to our `CHOReOSynt` tool by introducing a new RESTful service called `Synthesis Adapter Generator`. Furthermore, we propose a pipe-and-filter-based architecture of the generated adapters by using Spring Web Services and Spring Integration frameworks.

An explanatory example has been used to show two types of adaptation based on the Message Transformation pattern and the Message Aggregator pattern. The former plays a very important role by allowing the mediation of loose-coupling Message Producers and Message Consumers, which do not agree on a common data format. The latter is a type of Message Endpoint that receives multiple Messages and combines them into a single Message.

As future work, our plan is to fully implement the proposed extension and validate it on the case studies of the CHOReVOLUTION project.

Moreover, in order to achieve even more ambitious objectives within the CHOReVOLUTION project and to improve the applicability of the approach, we plan to extend it so as to deal with security aspects of the choreographies. This would allow for dealing with multiple services that belong to different security domains governed by different authorities, and use different identity attributes. This can be achieved by integrating EIPs with Security Patterns [35].

Acknowledgment. This research work has been supported by the Ministry of Education, Universities and Research, prot. 2012E47TM2 (project IDEAS - Integrated Design and Evolution of Adaptive Systems), by the European Union's H2020 Programme under grant agreement number 644178 (project CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), and by the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies).

References

1. European Commission: Digital Agenda for Europe - Future Internet Research and Experimentation (FIRE) initiative (2015)
2. Autili, M., Di Ruscio, D., Di Salle, A., Inverardi, P., Tivoli, M.: A model-based synthesis process for choreography realizability enforcement. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 37–52. Springer, Heidelberg (2013)
3. Autili, M., Di Salle, A., Tivoli, M.: Synthesis of resilient choreographies. In: Gorbenko, A., Romanovsky, A., Kharchenko, V. (eds.) SERENE 2013. LNCS, vol. 8166, pp. 94–108. Springer, Heidelberg (2013)
4. Salle, A.D., Inverardi, P., Perucci, A.: Towards adaptable and evolving service choreography in the future Internet. In: IEEE Services, pp. 333–337 (2014)
5. Autili, M., Inverardi, P., Tivoli, M.: Automated synthesis of service choreographies. IEEE Softw. **32**(1), 50–57 (2015)

6. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions - Printing 2011. Addison-Wesley Longman, Boston (2004)
7. Basu, S., Bultan, T.: Choreography conformance via synchronizability. In: Proceedings of WWW (2011)
8. Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Patrizi, F.: Automatic service composition and synthesis: the roman model. IEEE Data Eng. Bull. **31**(3), 18–22 (2008)
9. Hallé, S., Bultan, T.: Realizability analysis for message-based interactions using shared-state projections. In: Proceedings of FSE, pp. 27–36 (2010)
10. Pathak, J., Lutz, R., Honavar, V.: Moscoe: an approach for composing web services through iterative reformulation of functional specifications. Int. J. Artif. Intell. Tools **17**, 109–138 (2008)
11. Salaün, G.: Generation of service wrapper protocols from choreography specifications. In: Proceedings of SEFM (2008)
12. Poizat, P., Salaün, G.: Checking the realizability of BPMN 2.0 choreographies. In: Proceedings of SAC 2012 (2012)
13. Gössler, G., Salaün, G.: Realizability of choreographies for services interacting asynchronously. In: Arbab, F., Ölveczky, P.C. (eds.) FACS 2011. LNCS, vol. 7253, pp. 151–167. Springer, Heidelberg (2012)
14. Basu, S., Bultan, T., Ouederni, M.: Deciding choreography realizability. In: Proceedings of POPL. ACM (2012)
15. Güdemann, M., Poizat, P., Salaün, G., Dumont, A.: VerChor: a framework for verifying choreographies. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 226–230. Springer, Heidelberg (2013)
16. Salaün, G., Bultan, T., Roohi, N.: Realizability of choreographies using process algebra encodings. IEEE TSC **5**(3), 290–304 (2012)
17. Ouederni, M., Salaün, G., Bultan, T.: Compatibility checking for asynchronously communicating software. In: Fiadeiro, J.L., Liu, Z., Xue, J. (eds.) FACS 2013. LNCS, vol. 8348, pp. 310–328. Springer, Heidelberg (2014)
18. Basu, S., Bultan, T.: Automatic verification of interactions in asynchronous systems with unbounded buffers. In: Proceedings of ASE, pp. 743–754 (2014)
19. Güdemann, M., Poizat, P., Salaün, G., Dumont, A.: VerChor: a framework for verifying choreographies. In: Cortellessa, V., Varró, D. (eds.) FASE 2013 (ETAPS 2013). LNCS, vol. 7793, pp. 226–230. Springer, Heidelberg (2013)
20. Güdemann, M., Salaün, G., Ouederni, M.: Counterexample guided synthesis of monitors for realizability enforcement. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 238–253. Springer, Heidelberg (2012)
21. Shaw, M., Garlan, D.: Software Architecture - Perspectives on an Emerging Discipline. Prentice Hall, Upper Saddle River (1996)
22. Autili, M., Ruscio, D.D., Salle, A.D., Perucci, A.: Choreosynt: enforcing choreography realizability in the future Internet. In: Proceedings of FSE, pp. 723–726 (2014)
23. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proceedings of ESEC/FSE (2009)
24. Calvert, K.L., Lam, S.S.: Formal methods for protocol conversion. IEEE J. Sel. Areas Commun. **8**(1), 16 (1990)
25. Lam, S.S.: Correction to "protocol conversion". IEEE TSE **14**(9), 1376 (1988)
26. Vaculín, R., Sycara, K.: Towards automatic mediation of OWL-S process models. In: Proceedings of IEEE Web Services (2007)

27. Vaculín, R., Neruda, R., Sycara, K.: An agent for asymmetric process mediation in open environments. In: Kowalczyk, R., Huhns, M.N., Klusch, M., Maamar, Z., Vo, Q.B. (eds.) Service-Oriented Computing: Agents, Semantics, and Engineering. LNCS, vol. 5006, pp. 104–117. Springer, Heidelberg (2008)
28. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: Proceedings of ICSE (2003)
29. Passerone, R., Alfaro, L.D., Henzinger, T.A., Sangiovanni-Vincentelli, A.L.: Convertibility verification and converter synthesis: two faces of the same coin. In: Proceedings of ICCAD (2002)
30. Bennaceur, A., Issarny, V.: Automated synthesis of mediators to support component interoperability. IEEE TSE **41**(3), 221–240 (2015)
31. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Web, Web-Services, and Database Systems, pp. 221–237 (2002)
32. Do, H.H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: Proceedings of VLDB, pp. 610–621 (2002)
33. Massmann, S., Engmann, D., Rahm, E.: COMA++: results for the ontology alignment contest OAEI 2006. In: Proceedings of OM/ISWC (2006)
34. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
35. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns Integrating Security and Systems Engineering. Wiley, Verlag (2005)