On the Automated Synthesis of Enterprise Integration Patterns to Adapt Choreography-based Distributed Systems

Marco Autili Amleto Di Salle
Alexander Perucci Massimo Tivoli
Department of Information Engineering Computer Science and Mathematics
University of L'Aquila - ITALY
marco.autili@univaq.it amleto.disalle@univaq.it
alexander.perucci@graduate.univaq.it massimo.tivoli@univaq.it

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 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 [13]. 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 among them. It decouples 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 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 BPMN2 Choreography Diagrams, a role 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 in the choreography. Choreography roles can be specified by using different notations aiming at modeling different

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

José Proença and Massimo Tivoli (Eds.):
14th International Workshop on Foundations of
Coordination Languages and Self-Adaptive Systems (FOCLASA15)
EPTCS 201, 2015, pp. 33–47, doi:10.4204/EPTCS.201.3

views of the expected service behaviour, e.g., WSDL[2] for protocol signature specification, BPEL[3] or automata-based notations for protocol interaction specification.

In this paper we leverage the experience on the automatic synthesis of the choreography-based coordination logic for service-oriented systems that we have been doing so far within the EU CHOReOS project[4]. Then, being supported by the EU CHOReVOLUTION (follow-up) project[5], we report on the novel idea we are currently investigating to achieve choreography adaptation that, beyond coordination, permits to face the challenges posed by the heterogeneity of FI services.

In this direction, we propose a way to enhance the previous CHOReOS approach in [3, 2, 30, 1], and describes the preliminary steps we are undertaking within CHOReVOLUTION. The idea is to exploit Enterprise Integration Patterns [17] (EIP) so as to deal with a form of choreography adaptation. Specifically, distributed coordination logic is first automatically synthesized out of the BPMN2 choreography specification. This logic is concrete service independent, meaning that it is synthesized by considering the expected interaction protocol specified for the roles instead of the one of the concrete services. This allows our approach to realize separation of concerns, hence possibly reusing the synthesized coordination logic when (late) binding different concrete services to the choreography roles. Then, service adapters are synthesized (only when needed) in order to correctly realize service-role binding. After a set of concrete participant services have been selected as suitable (yet not perfectly matching) candidates to play the choreography roles, adapters let the protocol of concrete services match the one specified for the played roles. Specifically, an adapter solves protocol mismatches between a concrete service and the played role by exploiting EIP as adaptation primitives and by suitably composing them to realize the required adaptation logic.

As a preliminary work, in this paper, we consider a subset of the Message Routing EIP hence addressing some specific protocol mismatches. A more complete treatment of a greater number of EIP, and related mismatches, is left for future work.

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

2 Background notions on EIP

This section provides basic notions about the subset of EIP that our approach exploits so far to deal with protocol adaptation issues. As already introduced, to achieve adaptation, the protocol (i.e., operations signature plus interaction via messages exchange) of the concrete services may need to be adapted to the roles to be played in the BPMN2 choreography diagram given as input. 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.

EIP offer more than one style for integrating heterogeneous (possibly mismatching) applications, i.e., *File Transfer*, *Shared Database*, *Remote Procedure Invocation*, and *Messaging* [17]. 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).

[2]`http://www.w3.org/TR/wsdl`

[3]`https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm`

[4]`http://www.choreos.eu/`

[5]`http://www.chorevolution.eu/`

The Messaging approach uses the "pipes-and-filters" architectural style [32] as base for "streaming" messages between endpoints. The latter (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 of them dedicated to solve a particular integration aspect. Table 1 describes the subset of EIP that we consider in this paper.

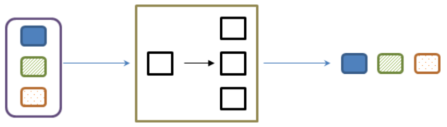
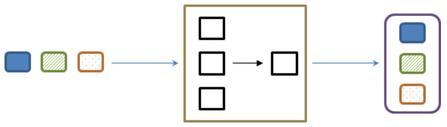
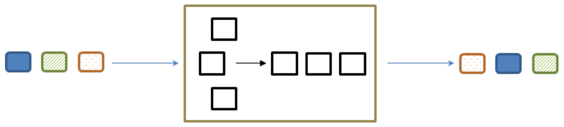
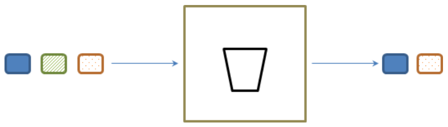
Name	**Description**	**Figure**
Splitter	It splits a message into several ones, and sends the resulting messages to be processed independently	
Aggregator	It receives multiple messages and combines them into a single message. It is stateful and it must buffer the messages to be aggregated and determine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

Table 1: Considered Message Routing EIP

The Aggregator, Splitter, Resequencer, and Message Filter belong to the class of Message Routing Patterns. This class of patterns is used to decouple a message source from the ultimate destination of the message following specific message routing policies as described in the table.

3 Explanatory Example

The 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 coordinating the activities 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[6]. The diagram 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.

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

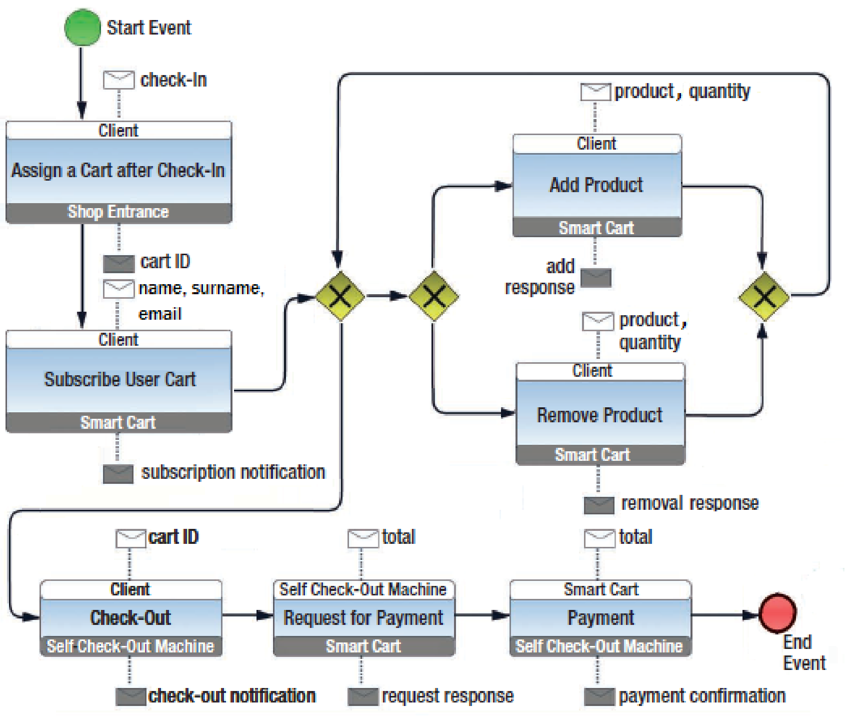


Figure 1: In-store Marketing and Sale choreography

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`.

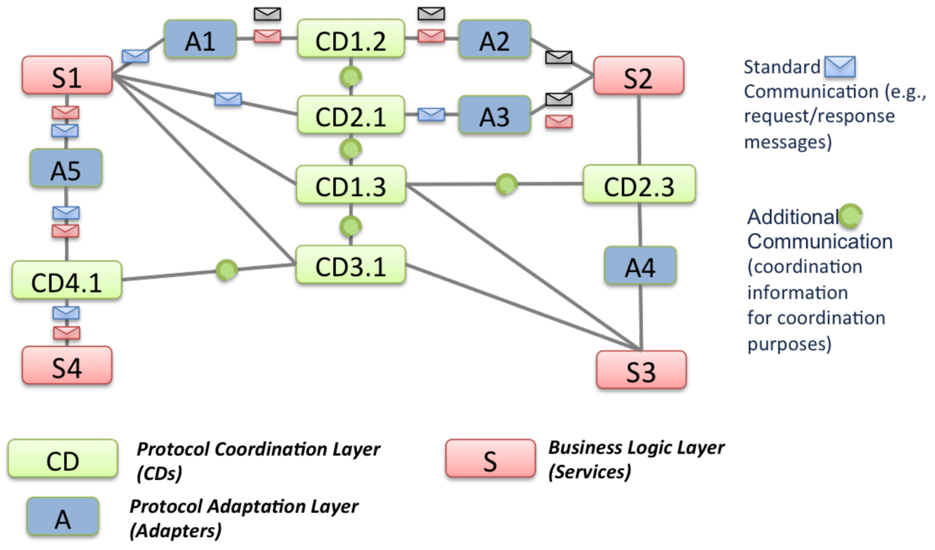
4 Method Description

The problem we want to solve with our synthesis method concerns the *automatic realizability enforcement* of service choreographies. It can be informally phrased as follows: given a choreography specification and a set of existing services, externally coordinate and adapt their interaction so to fulfill the collaboration prescribed by the choreography specification. Note that this problem is fundamentally different from the widely known *realizability check* (checks whether the choreography can be realized by implementing each participant so that it conforms to the played role) and *conformance check* (checks whether the set of services satisfies the choreography specification) problems [6, 28, 27, 14, 8, 15, 29, 7].

The method we propose distinguishes between *protocol coordination* and *protocol adaptation*.

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. For instance, the `Client` is allowed to perform the `Add Product` task to add products to the `Smart Cart` (see the topmost right side of Figure 1). However, after paying and before the end event, a "malicious" `Client` may attempt at adding products to the cart, without paying for them. In order to achieve correct protocol coordination, additional software entities are synthesized and interposed among the participant services. They are hereafter referred to as *Coordination Delegates* (CDs). As described in [5], CDs proxify and coordinate the participant services' interaction. When interposed among the services, according to the architectural style shown in Figure 2, CDs guarantee the collaboration specified by the choreography specification through distributed protocol coordination.

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 Figure 2). Each Adapter is generated to bridge/mediate the concrete service protocol in order to exactly match the abstract participant protocol. In other words, adapters realize correct service-role binding by solving possible interoperability issues (e.g., protocol mismatches). By exploiting EIP, the generated adaptation logic is realized as a composition of instances of message routing patterns that realize complex I/O data mappings. 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
A1
CD1.2
A2
S2
CD2.1
A3
A5
CD1.3
CD2.3
CD4.1
CD3.1
A4
S4
S3
Standard Communication (e.g., request/response messages)
Additional Communication (coordination information for coordination purposes)
CD
Protocol Coordination Layer (CDs)
A
Protocol Adaptation Layer (Adapters)
S
Business Logic Layer (Services)

Figure 2: Architectural style with adapters

Summing up, coordination and adaptation software entities are synthesized in order to: (i) proxify and coordinate the participant services' interaction to guarantee the global collaboration specified by the choreography (CDs), and (ii) adapt the concrete services' protocol to fit the protocol of the respective played roles (adapters).

As already introduced, an important aspect here is that the coordination logic performed by the CDs is *service-independent* since it is based on the expected protocol of the participants as specified by the choreography, rather than on the protocol of the 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, data incompatibilities at the service interface level, and interaction mismatches).

Section 4.1 presents an overall and high-level description of the method implementation. Then, by focusing on the novel contribution of this paper with respect to our previous work done in CHOReOS, Section 4.2 provides some details about the adapters generation step of our method.

4.1 Overview of the Method Implementation

Our previous work within CHOReOS [3, 2, 30, 1], and the related prototype implementation in the `CHOReOSynt` tool [4], concerns the automated synthesis of CDs out of the choreography BPMN2 specification. In order to automatically synthesize adapters, this paper advances our previous work and proposes an extension of `CHOReOSynt` by introducing a new RESTful service called `Adapter Generator` (see Figure 3).

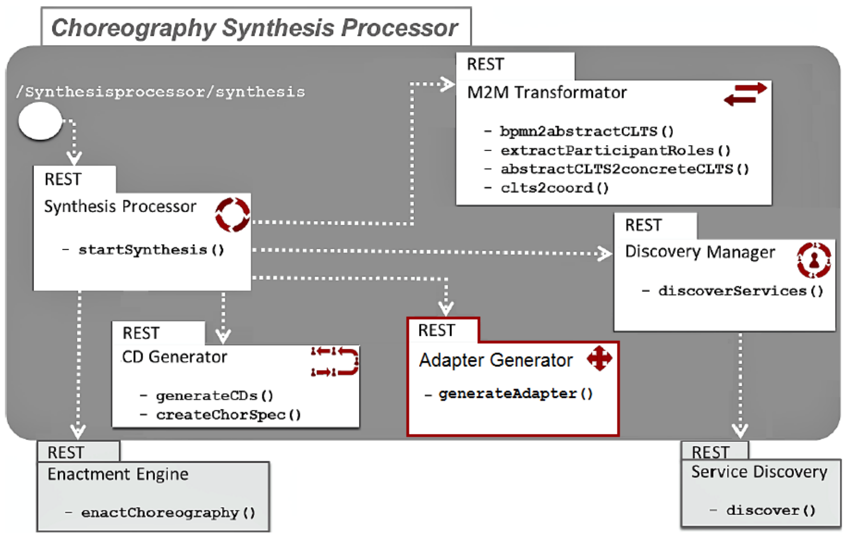


Figure 3: REST Architecture of the Synthesis Processor

By referring to Figure 3, the extension we propose allows for deriving both CDs and adapters starting from a BPMN 2.0 specification of the choreography, and a protocol specification of the concrete services selected as suitable participants. To this end, model transformations are employed and interoperation with the `Service Discovery` is required (not in the focus of this paper). Both CDs and adapters, when deployed by the `Enactment Engine` (not in the focus of this paper), allow for enacting the choreography by realizing the distributed coordination and adaptation logic between the discovered services.

The overall synthesis framework 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 transformations.

CD Generator – Starting from the choreography specification given as input (BPMN2 Choreography Diagram), the synthesis processor implements the approach formalized in [5] in order to generate the needed CDs through the operation `generateCD()`.

Synthesis Discovery Manager – The Synthesis process and the Discovery process interact each other to retrieve, from the service base, the candidate services that are suitable for playing the participant roles required by the choreography specification. That is, the services whose (provided and required) operations and protocol are semantically compatible (possibly, not perfectly matching) with the operations and protocol of the generated CDs.

Adapter Generator – For each discovered service, if needed, an adapter is automatically synthesized through the operation `generateAdapter()`. The aim of the adapter is to bridge/mediate the protocol of a discovered service with the one of the played role as reified by the corresponding CD.

4.2 Focusing on Adapters Generation

As already introduced, our method synthesizes adapters as WSs that implements a suitable composition of Message Routing EIP instances.

There are several frameworks and/or systems that implement/use EIP in order to integrate heterogeneous, and hence possibly mismatching, applications. We have chosen Spring Integration [7] because it implements most of the EIP, and it natively integrates with the Spring WSs project [8].

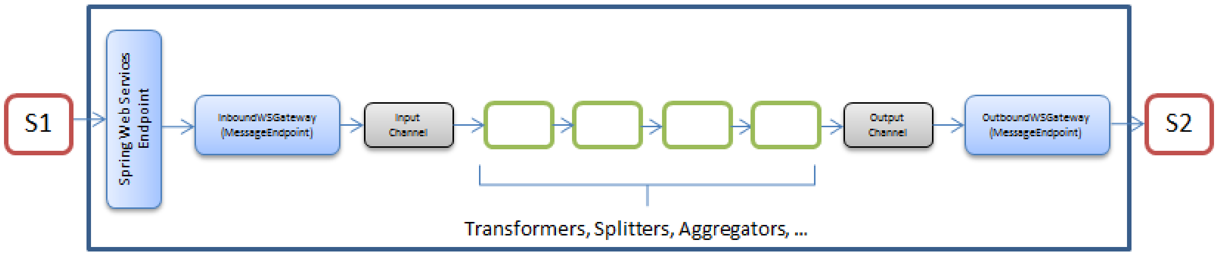

S1
Spring Web Services Endpoint
InboundWSGateway (MessageEndpoint)
Input Channel
Output Channel
OutboundWSGateway (MessageEndpoint)
S2
Transformers, Splitters, Aggregators, ...

Figure 4: Adapter architecture

Figure 4 describes the generic architecture of the generated adapters by using Spring WSs and Spring Integration. In particular, the Spring WSs `Endpoint` is the service that mediates the interaction between the service `S1` and the service `S2`. When `S1` invokes an operation *op1* by sending a message *m1*, the Endpoint receives the operation and put the message into the input channel by using *Inbound WS Gateways*. A sequential composition (chain) of suitable EIP is then generated depending on the interoperability issues discovered (e.g., protocol mismatches) between the two interacting services, e.g., a concrete service and a CD in our setting. An handler of the chain can be one or more *Message Routing EIP* as described in [17] and in Table 1.

In order to support the automated identification of interoperability issues, our method exploits a slightly modified version of the Strawberry tool [9]. In particular, Strawberry is used to automatically infer *data mappings* between different messages of two different WSs whose interaction needs to be mediated by an adapter. For instance, referring to our explanatory example, the two WSs could be the concrete `Client` service and the CD interposed between `Client` and `SmartCart`.

Strawberry is based on static data type analysis and testing. The former is used to analyze the type structure (e.g., the type structure of the XML Schema types of the messages in the WSDL of the considered WSs) of two messages belonging to the two different WSs. Based on the so far considered Message Routing EIP (Table 1), the adapters generation step reasons according to the following rules.

- **Message splitting and aggregation**. Given two different messages of two different services WS_1 and WS_2 with types t_1 and t_2, respectively, the aim of this analysis is to check whether, e.g., t_2 is a subtype of t_1 (denoted as $t_1 \preceq t_2$) according to the classical subtyping relation (i.e., signature inclusion): roughly, t_1 is contained in t_2. In our context, this means that either a message of type t_2 can be used to produce a message of type t_1 by means of message splitting or, vice versa, a message of type t_1 can be aggregated with other messages of WS_1 to produce a message of type t_2. Clearly, each of these other messages has to have a type t such that $t \preceq t_2$. Whether it is a case of message splitting or message aggregation depends on the "direction" of the communication that one wants to achieve by realizing the inferred data mappings. For instance, let t_2 (resp., t_1) be the type of

[7] `http://projects.spring.io/spring-integration/`
[8] `http://projects.spring.io/spring-ws/`

the input message of a required (resp., provided) operation, i.e., WS_2 behaves as the consumer and WS_1 as the provider. If $t_1 \preceq t_2$ then t_2 has to be split to produce t_1. Otherwise, if $t_2 \preceq t_1$, t_2 has to be aggregated with some other messages of WS_2 (for which t_1 is a subtype of their respective types) to produce t_1.

- **Message filtering**. It is worth to note that the data mappings inferred by Strawberry do not induce compositions of Splitter and Aggregator only. In fact, following the above discussion, for some pair of types t_1 and t_2, it can be the case that, e.g., no subtyping relation is defined for t_2 while t_1 has a subtyping relation with other message types of WS_2. This means that if WS_2 sends the message of type t_2 to WS_1, this message has to be consumed by exploiting a Message Filter.
- **Resequencing**. It can be the case that both $t_1 \preceq t_2$ and $t_2 \preceq t_1$ hold, denoted as $t_1 \equiv t_2$. In other words, t_1 and t_2 are the same type. Continuing the above discussion, let us suppose that WS_2 sends a sequence of messages whose types are $t_2^1 t_2^2 \ldots t_2^n$ to WS_1 while the latter expects to receive a sequence of messages whose types are $t_1^1 t_1^2 \ldots t_1^n$. Suppose also that, for all $i \in \{1, \ldots, n\}$, Strawberry inferred that $t_1^i \equiv t_2^{\pi(i)}$ where π is a permutation of $\{1, \ldots, n\}$, a Resequencer is needed in order to let WS_2 and WS_1 interact.

An important consideration here is that, in general, accounting for the messages' type structure only is not sufficient. In fact, one should check that two messages are also semantically correlated. The testing phase of Strawberry serves for this purpose. However, to perform testing with an acceptable accuracy, Strawberry requires the user to build an ad-hoc oracle. When testing cannot be performed, e.g., service providers do not offer a testing version of the services under analysis, the only thing that our method can do is to query the user (e.g., choreography designers, software architects) about confirming or not the semantic correlation of the data mappings inferred by Strawberry during the data type analysis. An alternative way to check message semantic correlation would be to exploit ontological information in a way similar to what is described in [18]. For instance, the use of ontological information will be indeed useful in the future since we plan to extend our method so to account also for other kinds of EIP (e.g., Message Transformation EIP like Content Filter or Content Enricher [17]).

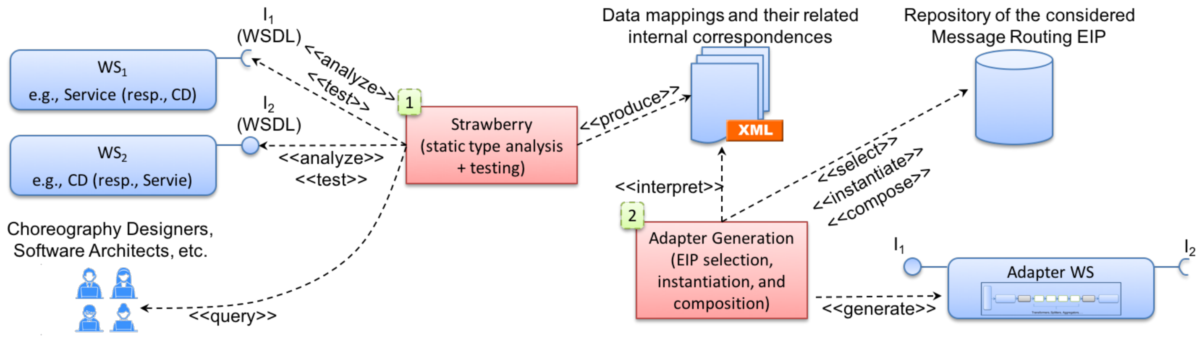


Figure 5: Adapters Generation Overview

Summing up, as shown in Figure 5, our adapters generation step can be overviewed as follows. Starting from (i) the specification of a consumer (resp., provider) participant service and (ii) the specification of the CD generated to coordinate the interaction between the consumer (resp., provider) role and the provider (resp., consumer) role, Strawberry produces a set of I/O data mappings and related internal data correspondences. As discussed above, these mappings allows for the detection of possible interoperability issues between the considered service and CD. These issues can arise when, e.g., the service required interface does not exactly match the CD provided one. This is done either completely automatically or semi-automatically by involving the user in the process. Then, an aptly coded model-to-code transformation is exploited in order to automatically generate the concrete adapter whose software architecture conforms to the one shown in Figure 4. This transformation interprets the elicited mappings and, according to the reasoning rules discussed above, select the required Message Routing EIP (among the considered ones), instantiate and compose them together to build the adapter.

5 Method at Work

In this section we describe the proposed enhancement at work based on the explanatory example shown in Section 3. We focus on the `Add Product` Choreography Task. The Client role models an abstract consumer service that requires the operation `addProduct(product, quantity)` to add a specified `quantity` of `product` to the cart. The Smart Cart role models an abstract provider service representing the cart, hence providing the operation `addProduct(product, quantity)`. As already introduced, from the choreography specification in Figure 1, the coordination delegate $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$ is synthesized. It coordinates (from outside) the interaction between the two services that will play the roles Client and Smart Cart respectively, in a way that the resulting collaboration realizes the specified choreography. Thus, $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$ is a prosumer service (both a provider and a consumer of service operations) that provides, as well as requires, the operation `addProduct(product, quantity)`.

Now, let us suppose that a concrete consumer service `Client` has been selected to play the Client role. `Client` requires three operations, `addProduct(product)`, `setQuantity(quantity)` and `setPromotionCode(promotionCode)`, in order to perform the `Add Product` Choreography Task. That is, when adding a specified `quantity` of `product` to the cart it can also take advantage of some discount by further specifying a `promotionCode`. Now, let us also suppose that a concrete provider service `SmartCart` has been selected to play the Smart Cart role. `SmartCart` provides two operations, namely `addItem(item)` and `setAmount(amount)`, which allow its clients for adding a specified `amount` of `item` to the cart.

5.1 Automated synthesis of data mappings

`Client` requires an interface that does not match the one provided by $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$. Similarly, `SmartCart` provides an interface that does not match the one required by $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$. Thus, two adapters must be synthesized, one between `Client` and $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$ and one between $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$ and `SmartCart`.

Strawberry is able to infer the following data mappings. Note that no mapping has been retrieved for the message `Client.setPromotion Code.setPromotionCodeRequest` containing `promotionCode`.

- `Client.addProduct.addProductRequest` $\preceq$ $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$`.addProduct.addProductRequest`
- `Client.setQuantity.setQuantityRequest` $\preceq$ $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$`.addProduct.addProductRequest`
- `SmartCart.setAmount.setAmountRequest` $\preceq$ $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$`.addProduct.addProductRequest`
- `SmartCart.addItem.addItemRequest` $\preceq$ $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$`.addProduct.addProductRequest`

Furthermore, the following internal data correspondences still concerning the inferred data mappings are also synthesized by Strawberry.

- `Client.addProduct.addProductRequest.product.id ->` $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.product.id`
- `Client.addProduct.addProductRequest.product.description ->` $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.product.description`
- `Client.setQuantity.setQuantityRequest.quantity ->` $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.quantity`
- $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.quantity -> SmartCart.setAmount.setAmountRequest.amount`
- $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.product.id -> SmartCart.addItem.addItemRequest.itemCode`
- $\texttt{CD}_{\texttt{Client\_SmartCart}}$`.addProduct.addProductRequest.product.description -> SmartCart.addItem.addItemRequest.descr`

As described in Section 4.2, these data correspondences are crucial for supporting the automated synthesis of the required adapter. That is, the concrete routing logic that the EIP instances corresponding to the inferred data mappings, and composed to form the adapter, have to implement.

The following listings show the type structure of the messages in the above mappings (for the sake of simplicity $\texttt{CD}_{\texttt{Client\_SmartCart}}$ is referred to as CD).

Listing 1: type structure of CD.addProduct.addProductRequest

```
<xsd:schema version="1.0" targetNamespace="http://choreosynth.disim.univaq.it/">
   <xsd:complexType name="product">
     <xsd:sequence>
        <xsd:element name="id" type="xsd:string"></xsd:element>
        <xsd:element name="description" type="xsd:string"></xsd:element>
     </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="quantity" type="xsd:int"></xsd:element>
</xsd:schema>
```

Listing 2: type structure of Client.addProduct.addProductRequest

```
<xsd:schema version="1.0" targetNamespace="http://choreosynth.disim.univaq.it/">
   <xsd:complexType name="product">
     <xsd:sequence>
        <xsd:element name="id" type="xsd:string"></xsd:element>
        <xsd:element name="description" type="xsd:string"></xsd:element>
     </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Listing 3: type structure of Client.setQuantity.setQuantityRequest

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

Listing 4: type structure of SmartCart.setAmount.setAmountRequest

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

Listing 5: type structure of SmartCart.addItem.addItemRequest

```
<xsd:schema version="1.0" targetNamespace="http://choreosynth.disim.univaq.it/">
   <xsd:complexType name="item">
```

```
    <xsd:sequence>
      <xsd:element name="itemCode" type="xsd:string"></xsd:element>
      <xsd:element name="descr" type="xsd:string"></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

5.2 Automated synthesis of adapters

As discussed in the previous section, our method exploits the data mappings inferred by Strawberry and the protocol specification of `Client`, $\mathrm{CD}_{\texttt{Client\_SmartCart}}$, and `SmartCart` in order to synthesize two adapters. They are $\texttt{Adapter}_1$ and $\texttt{Adapter}_2$ shown in Figure 6, which allow `Client` and `SmartCart` to communicate with $\mathrm{CD}_{\texttt{Client\_SmartCart}}$ despite protocol mismatches.

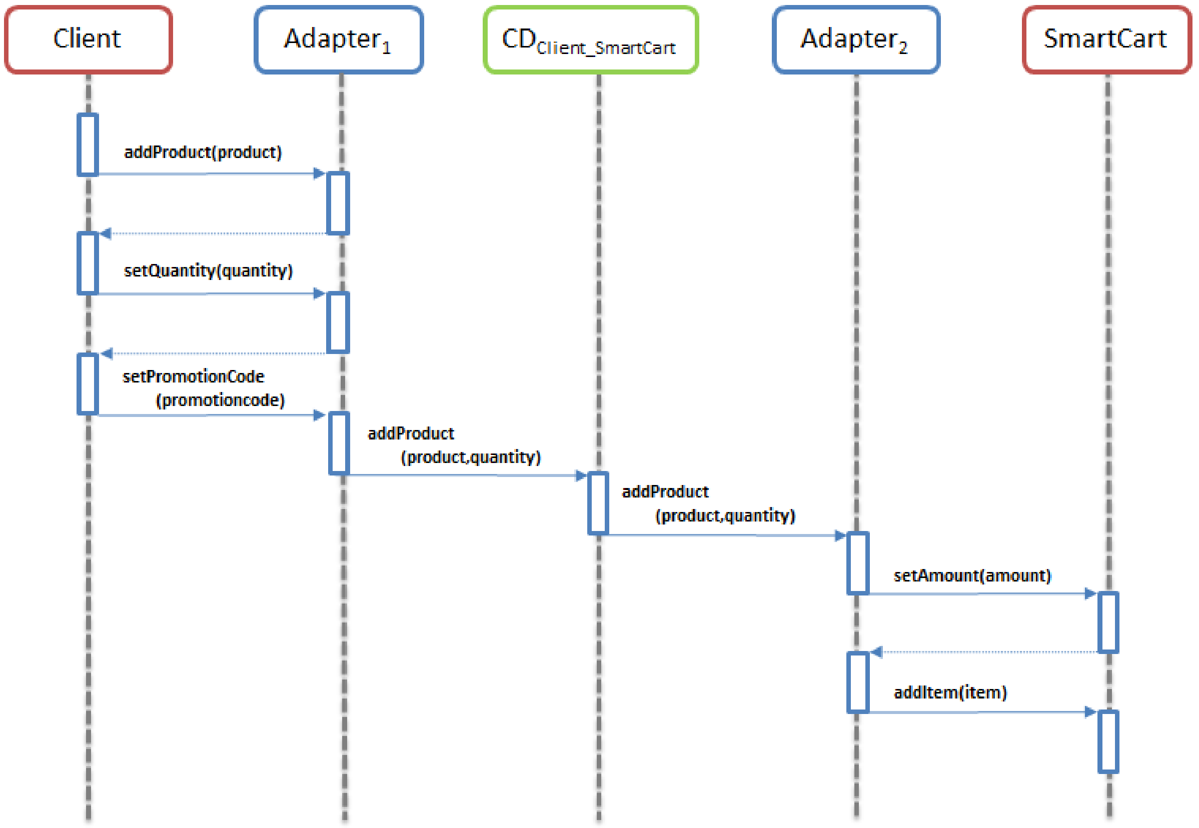


Figure 6: Adapter Example

$\texttt{Adapter}_1$ is generated by sequentially composing two EIP, namely Message Filter and Aggregator. The Message Filter takes as input from `Client` the three messages `addProductRequest`, `setQuantityRequest`, and `setPromotionCodeRequest`, and consumes `setPromotionCodeRequest` since there is no mapping inferred for it. This essentially means that the datum `promotionCode` is not needed to produce the data `product` and `quantity` contained in the `addProductRequest` message of $\mathrm{CD}_{\texttt{Client\_SmartCart}}$. Thus, at the end, Message Filter outputs `addProductRequest` and `setQuantityRequest`. Aggregator takes as input these two messages and merges them into one single message, hence producing the addProductRequest message of $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$.

$\mathtt{Adapter}_2$ is generated by sequentially composing two EIP, namely Splitter and Resequencer. Splitter takes as input addProductRequest from $\mathtt{CD}_{\mathtt{Client\_SmartCart}}$ and splits it in the two separate messages containing the data product and quantity, respectively. Resequencer takes these two messages as input and reorders them hence invoking the operation setAmount(amount) of SmartCart and then the operation addItem(item) of the same service. Note that, thanks to the inferred data mappings, the message setQuantityRequest (resp., addProductRequest) contains the data required to produce the message setAmountRequest (resp., addItemRequest). The code generated for $\mathtt{Adapter}_1$ and $\mathtt{Adapter}_2$ is available at http://choreos.disim.univaq.it.

6 Related

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 [10, 23]. 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 [34]. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

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 [33]. Although this formalizations supports modularization, automated synthesis is not treated, 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 [26]. In contrast to our method, their method needs as input a deadlock-free specification of the requirements that should be satisfied by the adapter, by delegating to the user the non-trivial task of specifying that.

In the context of *Reo connectors*, a number of related works are worth to be considered. The works described in [21, 22] discuss different approaches to extract choreography specifications from BPMN and UML diagrams. Automated synthesis of choreography specifications from scenario-based specification is addressed in [24]. The main focus of the works described in [20, 19] is on hybrid enforcement approaches.

Rahm et al. propose a catalog of criteria for documenting the evaluations of schema matching systems [11]. In particular, the authors discuss various aspects that contribute to the match quality obtained as the result of an evaluation. In [12] 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 [25] 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.

The approach described in [16] enforces choreography realizability by automatically generating monitors. Each monitor acts as a local controller for its peer. This approach obtains monitors by iterating equivalence-checking steps between two centralized models of the whole system. The adopted monitor concept is similar to our CD. However, our approach synthesizes CDs without producing a centralized model of the whole system, hence preventing state explosion.

7 Conclusion and Future Works

In this paper, we propose a way to enhance the approach to the automatic synthesis of choreography-based systems we previously proposed in the context of the EU FP7 CHOReOS project. We report the novel idea we are currently investigating within the EU H2020 CHOReVOLUTION project (a CHOReOS follow-up) to support choreography adaptation towards facing the challenges posed by the heterogeneity of Future Internet services. In particular, the idea is to automatically generate adapters by combining different EIP depending on a notion of I/O data mappings inference.

In order to automatically synthesize adapters we propose an extension of our `CHOReOSynt` tool introducing a new RESTful service called `Adapter Generator` and we define the architecture of the genereted adapters by using Spring WSs and Spring Integration frameworks. Referring to an explanatory example, we have shown two types of adaptation using some Message Routing Patterns, namely Message Filter, Aggregator, Splitter, and Resequencer.

The relevance of exploiting EIP to build adapters resides in the fact that the adapters generated by our method have a modular architecture since they are structured as a composition of independent EIP, each of them corresponding to the solution of a recurring protocol mismatch. This represents an overall advantage of our approach with respect to the work in the state of the art (Section 6). Because of the way adapters are modularly structured as a composition of fundamentals mediation primitives, the automatic generation of their actual code is viable and can be achieved with little effort.

As future work, in order to achieve the even more ambitious objectives of the CHOReVOLUTION project and to improve the applicability of the approach, we plan to extend it to deal with security issues concerning scenarios in which multiple services involved in a choreography belong to different security domains governed by different authorities and use different identity attributes that are utilized in their access control polices. This should be achieved by integrating EIPs with Security Patterns [31].

Furthermore, we plan to investigate how to support the automated synthesis of more complex adapters realized by combining other classes of EIP, e.g., Message Transformation Patterns such as Content Enricher, Content Filter, and Transformer. In general, this class of patterns provide a solution for achieving (semantic) interoperability between applications that communicate via message passing but rarely agree on a common data format. Message Transformation Patterns offer a general solution to possible differences in the data format of the exchanged messages. Thus, the ability to deal with these patterns in a systematic way would enable a finer form of adaptation concerning mismatches at the level of the semantics of the exchanged messages.

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] M. Autili, P. Inverardi & M. Tivoli (2015): *Automated Synthesis of Service Choreographies*. Software, IEEE 32(1), pp. 50–57, doi:10.1109/MS.2014.131.

[2] Marco Autili, Amleto Di Salle & Massimo Tivoli (2013): *Synthesis of Resilient Choreographies*. In: SERENE, LNCS 8166, doi:10.1007/978-3-642-40894-6_8.

[3] Marco Autili, Davide Ruscio, Amleto Di Salle, Paola Inverardi & Massimo Tivoli (2013): *A Model-Based Synthesis Process for Choreography Realizability Enforcement*. In: FASE, LNCS 7793, doi:10.1007/978-3-642-37057-1_4.

[4] Marco Autili, Davide Di Ruscio, Amleto Di Salle & Alexander Perucci (2014): *CHOReOSynt: enforcing choreography realizability in the future internet*. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014, pp. 723–726, doi:10.1145/2635868.2661667.

[5] Marco Autili & Massimo Tivoli (2015): *Distributed Enforcement of Service Choreographies*. In: Proceedings 13th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2014, Rome, Italy, 6th September 2014., pp. 18–35, doi:10.4204/EPTCS.175.2.

[6] Samik Basu & Tevfik Bultan (2011): *Choreography conformance via synchronizability*. In: Proc. of WWW '11, doi:10.1145/1963405.1963516.

[7] Samik Basu & Tevfik Bultan (2014): *Automatic verification of interactions in asynchronous systems with unbounded buffers*. In: ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014, pp. 743–754, doi:10.1145/2642937.2643016.

[8] Samik Basu, Tevfik Bultan & Meriem Ouederni (2012): *Deciding choreography realizability*. POPL, ACM, doi:10.1145/2103656.2103680.

[9] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione & Massimo Tivoli (2009): *Automatic Synthesis of Behavior Protocols for Composable Web-services*. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE '09, pp. 141–150, doi:10.1145/1595696.1595719.

[10] Kenneth L. Calvert & Simon S. Lam (1990): *Formal Methods for Protocol Conversion*. IEEE Journal on Selected Areas in Communications 8(1), doi:10.1109/49.46852.

[11] Hong Hai Do, Sergey Melnik & Erhard Rahm (2002): *Comparison of Schema Matching Evaluations*. In: Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, Erfurt, Germany, October 7-10, 2002, Revised Papers, pp. 221–237, doi:10.1007/3-540-36560-5_17.

[12] Hong Hai Do & Erhard Rahm (2002): *COMA - A System for Flexible Combination of Schema Matching Approaches*. In: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China, pp. 610–621, doi:10.1016/1287369.1287422.

[13] European Commission (2015): *Digital Agenda for Europe - Future Internet Research and Experimentation (FIRE) initiative*. Available at `https://ec.europa.eu/digital-agenda/en/future-internet-research-and-experimentation`.

[14] Gregor Gössler & Gwen Salaün (2012): *Realizability of Choreographies for Services Interacting Asynchronously*. In: FACS, LNCS 7253, pp. 151–167, doi:10.1007/978-3-642-35743-5_10.

[15] Matthias Güdemann, Pascal Poizat, Gwen Salaün & Alexandre Dumont (2013): *VerChor: A Framework for Verifying Choreographies*. In: FASE, LNCS 7793, pp. 226–230, doi:10.1007/978-3-642-37057-1_16.

[16] Matthias Güdemann, Gwen Salaün & Meriem Ouederni (2012): *Counterexample Guided Synthesis of Monitors for Realizability Enforcement*. In: Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings, pp. 238–253, doi:10.1007/978-3-642-33386-6_20.

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

[18] Paola Inverardi & Massimo Tivoli (2013): *Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns*. In: Proceedings of ICSE'13, doi:10.1109/ICSE.2013.6606546.

[19] Sung-Shik T. Q. Jongmans & Farhad Arbab (2013): *Global Consensus through Local Synchronization*. In: ESOCC Workshops, pp. 174–188, doi:10.1007/978-3-642-45364-9_15.

[20] Sung-Shik T. Q. Jongmans, Francesco Santini & Farhad Arbab (2014): *Partially-Distributed Coordination with Reo*. In: PDP, pp. 697–706, doi:10.1109/PDP.2014.19.

[21] Natallia Kokash & Farhad Arbab (2008): *Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems*. In: FMCO, pp. 21–41, doi:10.1007/978-3-642-04167-9_2.

[22] Natallia Kokash & Farhad Arbab (2013): *Formal Design and Verification of Long-Running Transactions with Extensible Coordination Tools*. IEEE T. Services Computing 6(2), pp. 186–200, doi:10.1109/TSC.2011.46.

[23] Simon S. Lam (1988): *Correction to "Protocol Conversion"*. IEEE Trans. Software Eng. 14(9), doi:10.1109/32.6181.

[24] Sun Meng, Farhad Arbab & Christel Baier (2011): *Synthesis of Reo circuits from scenario-based interaction specifications*. Sci. Comput. Program. 76(8), pp. 651–680, doi:10.1007/978-3-540-78743-3_12.

[25] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne & Katia P. Sycara (2002): *Semantic Matching of Web Services Capabilities*. In: The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings, pp. 333–347, doi:10.1007/3-540-48005-6_26.

[26] Roberto Passerone, Luca De Alfaro, Thomas A. Henzinger & Alberto L. Sangiovanni-Vincentelli (2002): *Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin*. In: ICCAD, doi:10.1145/774572.774592.

[27] Pascal Poizat & Gwen Salaün (2012): *Checking the Realizability of BPMN 2.0 Choreographies*. In: Proc. of SAC 2012, doi:10.1145/2245276.2232095.

[28] Gwen Salaün (2008): *Generation of Service Wrapper Protocols from Choreography Specifications*. In: Proc. of SEFM, doi:10.1109/SEFM.2008.42.

[29] Gwen Salaün, Tevfik Bultan & Nima Roohi (2012): *Realizability of Choreographies Using Process Algebra Encodings*. IEEE T. Services Computing 5(3), pp. 290–304, doi:10.1109/TSC.2011.9.

[30] Amleto Di Salle, Paola Inverardi & Alexander Perucci (2014): *Towards Adaptable and Evolving Service Choreography in the Future Internet*. In: 2014 IEEE World Congress on Services, SERVICES 2014, Anchorage, AK, USA, June 27 - July 2, 2014, pp. 333–337, doi:10.1109/SERVICES.2014.65.

[31] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann & Peter Sommerlad (2005): *Security Patterns Integrating Security and Systems Engineering*. John Wiley and Sons Ltd.

[32] Mary Shaw & David Garlan (1996): *Software architecture - perspectives on an emerging discipline*. Prentice Hall.

[33] Bridget Spitznagel & David Garlan (2003): *A Compositional Formalization of Connector Wrappers*. In: ICSE, doi:10.1109/ICSE.2003.1201216.

[34] Roman Vaculín, Roman Neruda & Katia P. Sycara (2008): *An Agent for Asymmetric Process Mediation in Open Environments*. In: SOCASE, doi:10.1007/978-3-540-79968-9_9.