An Architectural Style for the Development of Choreographies in the Future Internet

Dionysis Athanasopoulos[1], Marco Autili[2,*], Nikolaos Georgantas[3], Valérie Issarny[3], Massimo Tivoli[2] and Apostolos Zarras[4]

[1]Politecnico di Milano, Italy

[2]Università degli Studi di L'Aquila, Italy

[3]Inria Paris-Rocquencourt, France

[4]University of Ioannina, Greece

Abstract: Accounting for the challenges posed by the Future Internet (FI), we revisit the traditional definitions of component, connector and coordination protocol, and propose the CHOReOS Architectural Style (CAS) for the development of choreographies in the FI. Components enable leveraging the diversity of services that integrate in the FI as well as the ultra large service base envisioned for the FI. Connectors bring together the highly heterogeneous interaction paradigms that are now used in today's increasingly complex distributed systems and further support interoperability across heterogeneous paradigms. Coordination protocols foster choreography-based coordination for the sake of scalability, while preventing service interactions that would violate a specified choreography. A key aspect of CAS is to introduce novel abstractions for all its elements, which enable leveraging the wide diversity of the FI, in all its dimensions of scale, heterogeneity and mobility.

Keywords: Service Composition, Choreography, Synthesis, Distributed Systems, Service Discovery, Service-oriented Middleware.

1. INTRODUCTION

A software architecture style characterizes the types of components, connectors, and possibly configurations that serve building a given class of systems [1]. The style elements altogether specify the abstractions that need to be modeled, from design to implementation, as well as supported by the runtime to enact the target systems.

With in the European project CHOReOS (www.choreos.eu), we leveraged the Service, and related Service Oriented Architecture (SOA) and Service Computing paradigms for the Future Internet (FI). In other words, networked systems of the FI are exposed as (software) service providers and/or consumers (with service prosumer standing for a service that acts both as a consumer and a provider) in the networking environment. The service abstraction specifically serves characterizing functionalities that are provided and required in the networked environment so as to enable dynamic lookup and further binding between matching service consumer and producer. The SOA style may then be briefly defined as:

- components map to services, which may be refined into consumer, producer or prosumer services;
- connectors map to traditional client-service interaction protocols;
- configurations map to compositions of services through (service-oriented) connectors, i.e., choreography in the most general form, and orchestration as a specific composition structure that is commonly adopted in today's Internet.

The SOA style, as defined above, has led to many refinements, and further development of associated middleware technologies. Then, acknowledging the diversity of services to be composed, the Enterprise Service Bus (ESB) paradigm has been largely adopted in SOA, thereby introducing a connector type oriented toward interoperability across heterogeneous service-oriented middleware.

While the SOA style is well suited to support the development of Internet-based distributed systems, it is largely challenged by the Future Internet, aka FI, that poses new demands in terms of sustaining the following "ities" [2].

Scalability: the FI calls for novel service abstractions that shall serve characterizing and further locating all the component systems that will get networked at a massive scale. Scalability further advocates for weakly coupled interactions. Centralization shall be the exception, hence calling for choreography-based service composition.

*Address correspondence to this author at the Università degli Studi di L'Aquila, Italy; Tel: +39 0862 433186; Fax: +39 0862 433131;
E-mail: marco.autili@di.univaq.it

© 2014 Avanti Publishers

Heterogeneity: service abstractions are already much heterogeneous in today's Internet, where both SOAP and REST-full services co-exist [3]. They will be even more so as highly heterogeneous services, ranging from Business to Thing-based ones, will be integrated within the FI. The heterogeneity of networked services further leads to account for various connector types implementing different interaction paradigms, as well as for interoperability across interaction paradigms.

Mobility: mobility directly affects all the architectural elements, as it calls for system architectures whose components may be highly volatile and hence requires dealing with service substitution within composition.

Awareness & adaptability: it must be considered that services that get composed have not been designed in conjunction and thus may exhibit erroneous behavior when choreographed.

Purpose of the Paper: By discussing part of our work done within the CHOReOS project, the next three sections define the CHOReOS Architectural Style (CAS) that we have elaborated to address the above challenges, and embed a comparison with related work. Indeed, the CAS definition has been one of the main outcomes of the CHOReOS project. A complete formal definition of CAS is provided in [4], while in this paper, we summarize the CAS definition and do not go into formal details. However, we strive to keep a clear description of the key concepts and notions constituting the CAS.

Specifically, Section II defines CHOReOS service components in a technology-agnostic way and allows for abstracting Business as well as Thing-based services. The main innovation of the section comes from the definition of new abstractions for services that enables hierarchical structuring and, hence, scalable abstraction-oriented service bases.

Section III tackles the issue of interoperability across heterogeneous interaction paradigms, from strongly to weakly coupled ones. It introduces: (i) base connector types that abstract core interaction paradigms implemented by state-of-the-art middleware solutions, and (ii) a Generic Application (GA) connector type that realizes interoperability in a way similar to a service bus, but across interaction paradigms. The main contribution of the section lies in enabling cross-paradigm interoperability, while preserving the behavioral (at middleware-level) semantics of the connected components.

Section IV focuses on the formalization of the notion of CHOReOS coordination protocol that abstracts choreography behavior. The section defines a specific configuration to be imposed on the choreography-based system to suitably coordinate the composed services, and further enable the automated synthesis of the coordination protocol. The main novelty of the section derives from the efficient production of a decentralized choreographer, while avoiding those interactions that do not belong to the set of interactions modeled by the choreography specification, i.e., undesired interactions.

Section V concludes and discusses possible future work.

2. CHOREOS COMPONENTS: ABSTRACTING FI SERVICES

Dealing with the impact of the FI challenges on service-oriented components is one of the main issues dealt with in CAS. Regarding heterogeneity and mobility, CAS provides a unified formal definition of services that abstracts details related to the particular paradigms, standards and technologies on which services are based and further accounts for the specificities of Things-based services. An earlier attempt in this direction was the NEXOF reference architecture [5]. However, the NEXOF definition of services is rather informal and it does not account for services' behavior. Considering scalability, awareness & adaptability, CAS formally defines the concept of service abstractions. They represent groups of alternative services that provide similar functional and non-functional properties through different interfaces. Previous attempts in this direction were represented by semantic descriptions of services such as DAML-S (*www.ai.sri.com/daml/services/owl-s/*) and its successor OWL-S (*www.w3.org/Submission/OWL-S/*). Differently from them, CAS takes one step further since the CHOReOS service abstractions can be hierarchically structured.

2.1. Services in the FI

From an architectural point of view, the main features of components include the components' interfaces, semantics, constraints and properties [6]. In CHOReOS, we consider these main features as the basic constituents of the CHOReOS definition of services. Still, such a definition shall account for the heterogeneity of services to be aggregated in the FI, while acknowledging that services are essentially (if not uniquely) Web-based at the FI level. However, Web-

based Services are called to evolve to face the FI challenges when diverse interaction paradigms must be explicitly accounted for. Moreover, according to the Internet of Things overall view [7], in the near future it is expected that an ultra large number of devices will encompass computing and communication capabilities that will allow them to interact with their surrounding environment (including the physical world) and the inverse.

The SOAP and RESTful paradigms [3] offer two different alternatives for the realization of Thing-based services. Specifically, RESTful services are considered as the preferable approach for the realization of Thing-based services in the cases where the functionalities offered by Things are quite simple and atomic, in the sense that there are no complex conversations involved between Things and their surrounding environment [8].

On the other hand, for Things that offer more complex functionalities, SOAP services are considered as a better option. Nevertheless, Things in general have limited computational/communication capabilities and resources [8]. Consequently, it is expected that Thing-based services would comply with a limited subset of standards that can be supported by Things [9].

Based on the above discussion, within CAS we consider paradigm-independent definitions of CHOReOS components, which span Business and Thing-based services. These definitions concern the notions of service type, interface, operation, and service instance.

A service type specifies: (i) the service profile, as a user-intuitive explanation of the service (e.g., see DAML/OWL-S/SAWSDL), (ii) the service style, which can be SOAP or RESTful, and (iii) a set of interfaces that specify the functionalities provided by the service.

A service interface defines: (i) a set of operations, that correspond to different functionalities provided through the interface, (ii) a (optional) behavioral specification which, independently of specific standards (e.g., BPEL, WSCL), can be seen abstractly as a Labeled Transition System (LTS), and (iii) a (optional) set of constraints related to requirements over the environment where the interface functionalities execute.

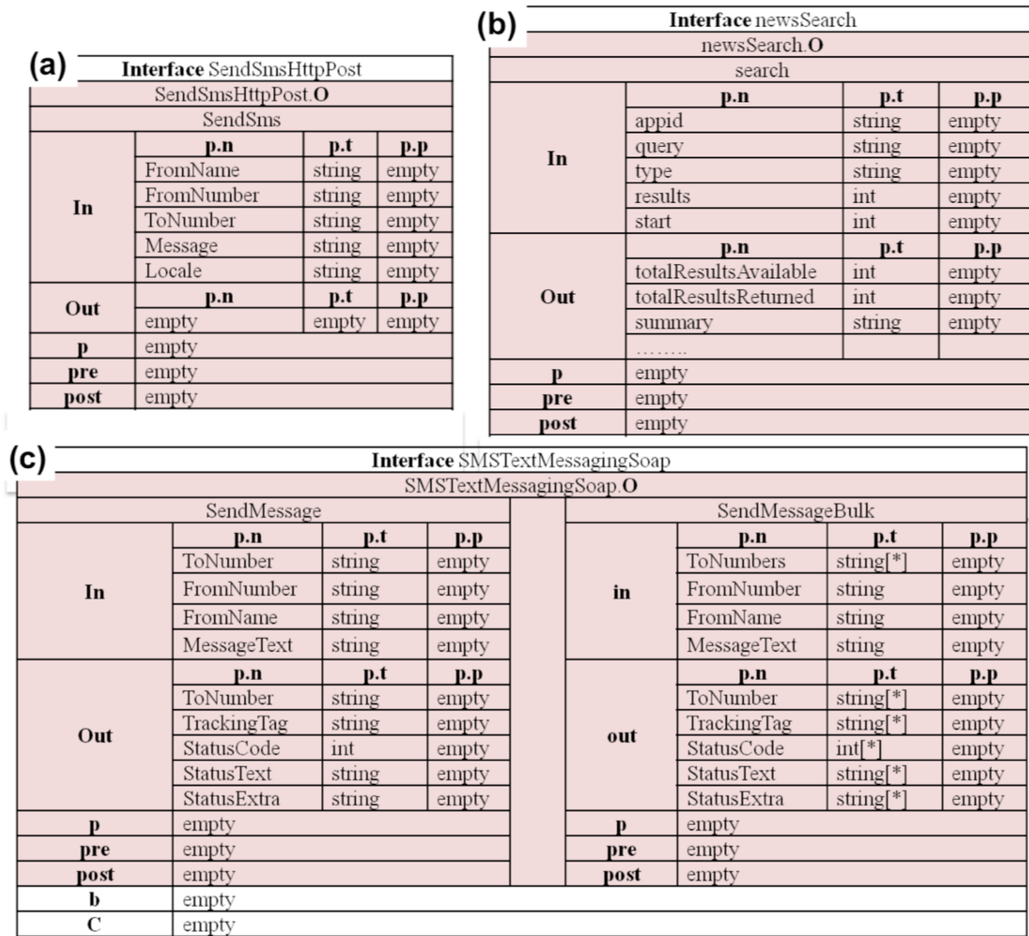
An operation has input/output parameters and can optionally specify pre- and post-conditions.

A service instance implements some interfaces, has an endpoint address plus optionally a behavioral description (e.g., an LTS) of its interaction protocol, and a (optional) description of non-functional properties given in terms of quality indicators, and associated measures. We can distinguish between runtime quality indicators (e.g., reliability, availability, reputation, price, performance), design quality indicators (e.g., different kinds of cohesion for service interfaces), and physical properties that typically characterize Thing-based services. In the following, we use a concrete example to briefly illustrate the notions given above and refer to [4] for a complete formalization of them.

Browsing the RemoteMethods registry (*www.remotemethods.com/*), we found various services that allow sending SMS messages to mobile phones. SMS-TXT is such a service that follows the SOAP paradigm. In the WSDL specification of the service interface we have a single operation named SendSms. The operation accepts as input message, five string parameters. The output message of the operation is empty. With respect to CAS, the service interface is specified as showed in the tabular representation given in Figure **1a**. The interface does not include profile specifications since there is no such information available in the registry where the service was found. For the same reason, there are no behavioral specifications and constraints.

An example of the CHOReOS representation of a RESTful service interface is given in Figure **1b**. The example is based on the Yahoo news search application [10]. This application consists of a single resource that provides a single method called search. Consequently, in the CHOReOS representation, we have an interface that defines a corresponding operation.

Finally, another example of an SMS service interface, GlobalSMSPro, is given in Figure **1c**. It also follows the SOAP paradigm and offers a more complex interface. As in the previous case, the interface does not include information regarding semantics, behavior, constraints and non-functional properties, since there is no such information available in the registry where the service was found. The interface consists of 6 operations. The SendMessage and SendMessages Bulk operations that are given in Figure **1c** are the ones that actually send SMS messages to mobile phones, while the rest of the operations (omitted in Figure **1c** for reasons of simplicity) serve for monitoring the status of SMS messages, or finding information



(a)

Interface SendSmsHttpPost			
SendSmsHttpPost.O			
SendSms			
In	p.n	p.t	p.p
	FromName	string	empty
	FromNumber	string	empty
	ToNumber	string	empty
	Message	string	empty
	Locale	string	empty
Out	p.n	p.t	p.p
	empty	empty	empty
p	empty		
pre	empty		
post	empty		

(b)

Interface newsSearch			
newsSearch.O			
search			
In	p.n	p.t	p.p
	appid	string	empty
	query	string	empty
	type	string	empty
	results	int	empty
	start	int	empty
Out	p.n	p.t	p.p
	totalResultsAvailable	int	empty
	totalResultsReturned	int	empty
	summary	string	empty
	……..		
p	empty		
pre	empty		
post	empty		

(c)

Interface SMSTextMessagingSoap			
SMSTextMessagingSoap.O			
SendMessage			
In	p.n	p.t	p.p
	ToNumber	string	empty
	FromNumber	string	empty
	FromName	string	empty
	MessageText	string	empty
Out	p.n	p.t	p.p
	ToNumber	string	empty
	TrackingTag	string	empty
	StatusCode	int	empty
	StatusText	string	empty
	StatusExtra	string	empty
p	empty		
pre	empty		
post	empty		

SendMessageBulk			
in	p.n	p.t	p.p
	ToNumbers	string[*]	empty
	FromNumber	string	empty
	FromName	string	empty
	MessageText	string	empty
out	p.n	p.t	p.p
	ToNumber	string[*]	empty
	TrackingTag	string[*]	empty
	StatusCode	int[*]	empty
	StatusText	string[*]	empty
	StatusExtra	string[*]	empty
p	empty		
pre	empty		
post	empty		

b	empty
C	empty

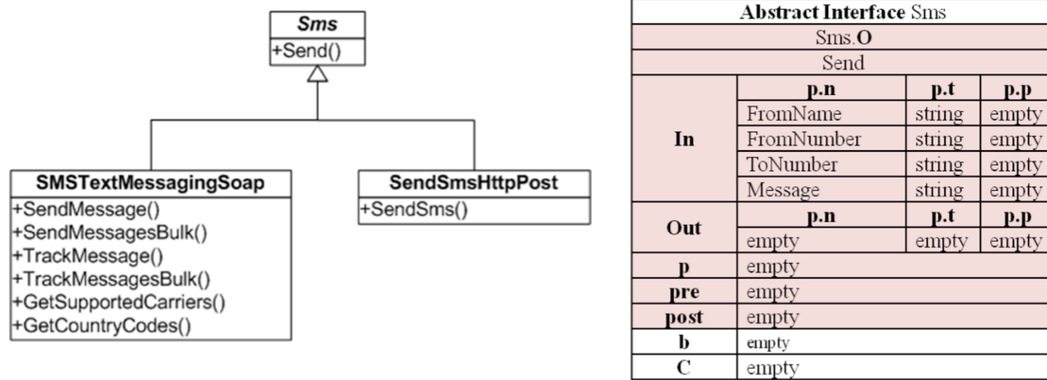
Figure 1: SMS-TXT (**a**), Yahoo newSearch (**b**), GlobalSMSPro (**c**).

concerning different country codes, network standards, etc.

2.2. Abstractions for FI Services

CAS provides formally grounded service abstractions that can be hierarchically structured. Moreover, CAS distinguishes between two different types of interrelated service abstractions: *functional (resp., non-functional) abstractions that represent groups of services that offer similar functional (resp., non-functional) properties.* The CAS service abstractions are considered as core elements that are mined from services that become available over time and assist the task of service discovery and the subsequent tasks of service composition and adaptation. The discovery process amounts to: (i) finding a functional abstraction that satisfies any or most of the functional requirements; (ii) returning the matching functional abstraction along with the services that are described by it. It may return too many functionally equivalent services, thus complicating the selection of the appropriate service. This issue becomes even more important considering that the actual service selection may take place at runtime. Hence, we need to provide extra selection criteria; we do this by utilizing the non-functional properties of services (e.g., response time, availability, throughput etc.). So, the selection of an actual service from the set of services elicited in the previous step consists of: (i) browsing or searching for the suitable non-functional abstraction that meets certain non-functional requirements; (ii) selecting any of the services of the selected non-functional abstraction.

Getting back to our example of Figure **1**, an interesting observation is that although these two SMS services come from two different providers their interfaces offer at least a pair of very similar operations. Specifically, the SendSms operation of SMS-TXT and the SendMessage operation of GlobalSMSPro realize semantically compatible functionalities (i.e., sending an SMS message) and further have very similar input parameters. This observation points out an opportunity for defining a service abstraction that represents these two services.



Abstract Interface Sms			
Sms.**O**			
Send			
In	**p.n**	**p.t**	**p.p**
	FromName	string	empty
	FromNumber	string	empty
	ToNumber	string	empty
	Message	string	empty
Out	**p.n**	**p.t**	**p.p**
	empty	empty	empty
p	empty		
pre	empty		
post	empty		
b	empty		
C	empty		

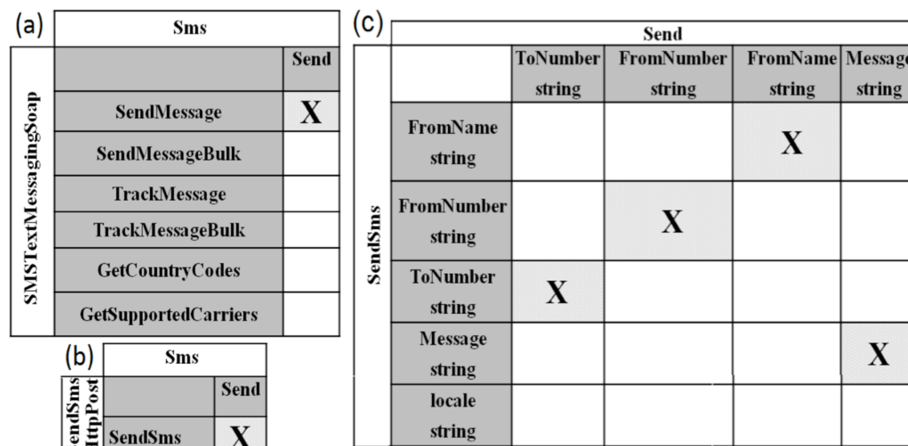
Figure 2: Example of a service functional abstraction.

A possible interface for this abstraction is given in Figure **2**. The abstract service interface that represents the interfaces of the two services comprises a single operation named Send and requires 4 string parameters. The mapping of this abstract interface to the interfaces of the two services is rather straightforward and is given in Figure 3**a** and **b**. Finally, Figure **3c** gives an example of a mapping between the input parameters of the abstract operation Send and the input parameters of the SendSms operation.

Although the definition of a functional abstraction seems trivial for the two services of our example, in the general case, where the given set of available services is large and the services much more complex, this task is challenging. To deal with this challenge in [11] we proposed a hierarchical clustering approach for mining functional service abstractions from sets of existing services, while in [12] we elaborated on a clustering approach that mines non-functional service abstractions. In the context of CHOReOS, we employed the concept of service abstractions for indexing service specifications in a service base and we demonstrated the benefits of this indexing method in terms of service querying execution time in large service base instances with up to 10^7 service specifications [12]. The main outcome was that querying over service abstractions is much (80% to 90% in our experimental settings) faster than querying over concrete service specifications. On the less bright side, the query execution time speedup may come at the expense of recall [13]. In the context of CHOReOS, we further employed the concept of service abstraction for the development of a reflective service adaptation mechanism that allows to substitute services that are represented by the same service abstraction [13].

3. CHOREOS CONNECTORS: INTEROPERABILITY IN THE FI

Complex distributed applications in the FI (including the Internet of Things constituent) will be to a large extent based on the open integration of extremely heterogeneous systems, such as lightweight



(a)

SMSTextMessagingSoap	Sms
	Send
SendMessage	X
SendMessageBulk	
TrackMessage	
TrackMessageBulk	
GetCountryCodes	
GetSupportedCarriers	

(b)

SendSms HttpPost	Sms
	Send
SendSms	X

(c)

SendSms	Send: ToNumber string	FromNumber string	FromName string	Message string
FromName string			X	
FromNumber string		X		
ToNumber string	X			
Message string				X
locale string				

Figure 3: Operation (**a,b**) and parameter (**c**) mappings.

embedded systems (e.g., sensors, actuators and networks of them), mobile systems (e.g., smartphone applications), and resource-rich IT systems (e.g., systems hosted on enterprise servers and Cloud infrastructures). These heterogeneous system domains differ significantly in terms of interaction paradigms, communication protocols, and data representation models, which are most often provided by supporting middleware platforms. In particular with regard to middleware-supported interactions, the client/server (CS), publish/subscribe (PS), and tuple space (TS) paradigms are among the most widely employed ones today, with numerous related middleware platforms, such as: Web Services, Java RMI for CS; WS-Eventing, JMS, SIENA for PS [14]; and JavaSpaces, Lime for TS [15]. In light of the above, the connector types associated with CAS shall: (i) leverage the diversity of interaction paradigms associated with today's and future distributed systems, as well as (ii) enable cross-paradigm interaction to sustain interoperability in the highly heterogeneous FI.

As surveyed in [16,17], existing cross-domain interoperability efforts are based on, e.g., bridging communication protocols [18], wrapping systems behind standard technology interfaces [19], and/or providing common API abstractions [20]. In particular, such techniques have been applied by the two core system integration paradigms adopted by CHOReOS, that is, SOA and ESB [18]. However, state of the art interoperability efforts commonly cover part of the heterogeneity issues (regarding interaction, communication, data) and are applicable to specific cases [21]. In particular, existing solutions do not or only poorly address interaction paradigm interoperability. Specifically, SOA and ESB are primarily based on the CS paradigm. Even if extensions have been proposed, such as event-driven SOA or ESB supporting the PS paradigm [18], these remain partial. This means that systems integrated *via* SOA and ESB solutions have their interaction semantics transformed to the CS paradigm. Then, potential loss of interaction semantics can result in suboptimal or even problematic system integration.

To overcome the limitation of today's ESB-based connectors for cross-domain interoperability in the FI, we introduce a new connector type, called *Generic Application (GA) connector*. GA relies on the service bus paradigm, but, in contrast to classical ESB connectors, it particularly addresses interaction paradigm interoperability by paying special attention to the preservation – as much as possible – of semantics when bridging across heterogeneous paradigms. GA is based on the abstraction and semantics-preserving merging of the common high-level features of base interaction paradigms. This solution serves rethinking the typical SOA- and ESB- based composition of heterogeneous distributed systems so as to meet the requirements of the FI.

Figure **4** depicts our overall approach to interoperability across interaction paradigms. While networked services rely on legacy protocols and hence use their associated API to interact with their environment, these legacy protocols are mapped onto corresponding primitives of the GA connector. Then, internally to the GA connector, possible architectural mismatches are solved based on the pairwise matching of GA's input primitives with GA's output primitives.

3.1. Base Connector Types

Before introducing GA, we provide a brief description of the base connectors.

The CS *connector type* integrates a wide range of semantics, covering both the *direct* (i.e., *non queue-based*) *messaging* and *remote operation call* paradigms. It also enables both blocking and non-blocking reception semantics. CS imposes *space coupling* between the two interacting entities, i.e., at least the sending entity must know the receiving entity

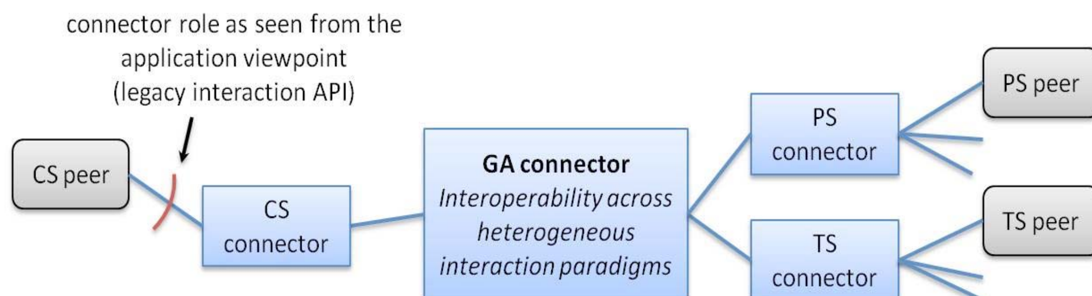


Figure 4: GA-based connector interoperability.

and hold a reference of it, as well as *time coupling*, i.e., both entities must be connected at the time of the interaction. A *message* carrying data is sent directly from the sending entity to the receiving entity.

In the PS paradigm, multiple peer entities interact *via* an intermediate *broker* entity. Publishers produce *events* carrying data, which are received by peers that have previously subscribed for receiving the specific events. The *PS connector type* abstracts in a comprehensive way the different types of publish/subscribe systems, such as *queue*-, *topic*- and *content-based systems* [22]. It further enables rich reception semantics: synchronous pull by the subscriber, which may be blocking or non-blocking, or asynchronous push by the broker. PS enables *space decoupling* between interacting peers: peers do not know each other; with the exception of queue-based PS, where at least the publisher holds a reference of the subscriber. Moreover, PS enables *time decoupling*: peers do not need to be present at the same time, subscribers maybe disconnected at the time of the interaction; they can receive the pending events when reconnected. The broker maintains an event until all related subscribers have received it or until the event expires.

In the TS paradigm abstracted by the *TS connector type*, multiple peer entities interact *via* a *shared data space*. Data take the form of *tuples*; a tuple is an ordered list of typed elements. Peers can post data into the space and also synchronously retrieve data from it, either by taking a copy or removing the data. Data are retrieved by matching based on a tuple template, which may define values or expressions for some of the elements. *Bulk* primitives [15] enable retrieving all the matching tuples. Peers can alternatively set up a callback function that is triggered asynchronously by the data space when matching data appear. This call does not carry the data; the action of taking or reading the data should be executed by the peer. TS enables both *space and time decoupling* between interacting peers. Nevertheless, TS has a number of specifics. Peers have access to a single, commonly shared copy of the data. Also, peers do not need to subscribe for data, they can retrieve data spontaneously and at any time. The data space maintains data until they are removed by some peer or until the data expire. Additionally, concurrency semantics of the data space are non-deterministic: among a number of peers trying to access the data concurrently, the order is determined arbitrarily.

3.2. Generic Application Connector Type

We now introduce the CHOReOS Generic Application (GA) connector type. Our objective is to devise a single generic connector that comprehensively incorporates the end-to-end interaction semantics of application entities that employ any of the three above base, i.e., CS, PS and TS, middleware connectors. This leads us to identify two principal high-level primitives for the GA connector enabling service interaction with the environment:

1) A *post()* primitive employed by a peer for sending data to one or more other peers (i.e., production of information), where data may represent CS messages, PS events, or TS tuples. For example, a PS *publish()* primitive can be abstracted by a *post()*.

2) A *get()* primitive employed by a peer for receiving data (i.e., consumption of information).

Then, following the definition of the base connector types, a producer/post - consumer/get end-to-end interaction may be characterized by one of the following three types of coupling: *strong coupling* corresponding to the CS paradigm; *weak coupling* for PS; or *very weak coupling* for TS.

GA connector API: The complete set of GA primitives is given at the top of Figure **5**, where we have applied a pseudo C-like syntax. They are defined as follows:

- *post()* produces *data* in the networking environment according to the semantics set by *coupling* (i.e., *strong*, *weak*, *very weak* as defined above). We introduce the explicit scoping parameter *scope* to generalize addressing for the different types of coupling. Hence, *scope* may represent: the address and message/operation qualifier of a CS receiving entity; the address and event qualifier of a PS broker; or the address and tuple qualifier for a TS tuplespace. *lease* determines the lifetime of the *data* before their expiration.

- *set_get()* sets up reception resources at the GA connector. *handle* is returned by the connector and serves identifying the reception setup.

- *get_async()* enables asynchronous reception of *data via* a *callback* called by the connector when *data* are ready. *scope* may additionally be returned by the *callback* to identify the sending entity.

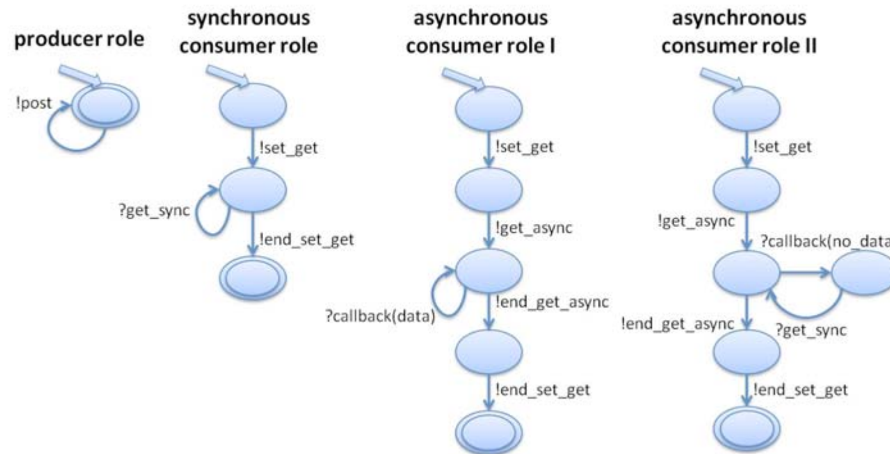


Figure 5: GA connector API (top) and interaction semantics (bottom).

post(coupling, scope, data, lease)
set_get(coupling, scope, *handle)
get_async(handle, *callback(*scope, *data))
get_sync(handle, policy, *data, timeout)
end_get_async(handle)
end_set_get(handle)

- *get_sync()* executes synchronous (blocking) reception of *data* with a *timeout*. We introduce *policy* to specialize this primitive in the case of very weak coupling (TS paradigm). Thus, *policy* may be *remove*, *copy*, *remove_all*, *copy_all*, which correspond to the various TS data retrieval fashions. For strong or weak coupling, *policy* is *remove* by default.
- *end_get_async()* ends asynchronous reception.
- *end_set_get()* closes a reception setup.

GA interaction semantics: The behaviors of the GA connector roles are specified with a graphical LTS representation at the bottom of Figure **5**. Precisely, the Figure specifies the behaviors that enable to produce and consume information to/from the networked environment using the connector API. In particular, we note that the LTS actions are labeled according to the operations of the connector API that they abstract, and that ! (resp. ?) denotes an output (resp. input action). The production of information is synchronous, although space and time decoupling with the consumer is possible, depending on the actual parameters of *post*. On the other hand, we distinguish between synchronous and asynchronous consumption of information. Additionally, we identify two forms of asynchronous consumption, depending on whether the consumption is in the *push* (role I) or *pull* (role II) mode.

The above presented GA connector type provides an abstract union of the base CS, PS and TS connector types, thus preserving by construction their interaction semantics. We employ this desirable feature for enabling interoperability across the CS, PS and TS paradigms in the next section.

3.3. Interoperability Across Interaction Paradigms

The GA connector *glue* process must ensure proper coordination of semantically matching producers and consumers despite heterogeneity of the actual interaction paradigms executed by the services. In other words, we want to enable a *get* and a *post* action of GA to coordinate if they *semantically match* in terms of the application semantics they carry although mismatching from the standpoint of the interaction paradigms they abstract. By its construction, as presented in the previous section, GA merges in a semantics-preserving way the common high-level features of base interaction paradigms. Nevertheless, certain semantics are different across paradigms, and, in this case, a mapping is required that respects the individual semantics as much as possible. The principal semantic differences between paradigms are concentrated in their applied *coupling*. Hence, the GA glue shall realize the necessary mediation between heterogeneous coupling semantics for a piece of data to be properly exchanged between the sender (*poster*) and its semantically matching receiver(s) (*getter(s)*).

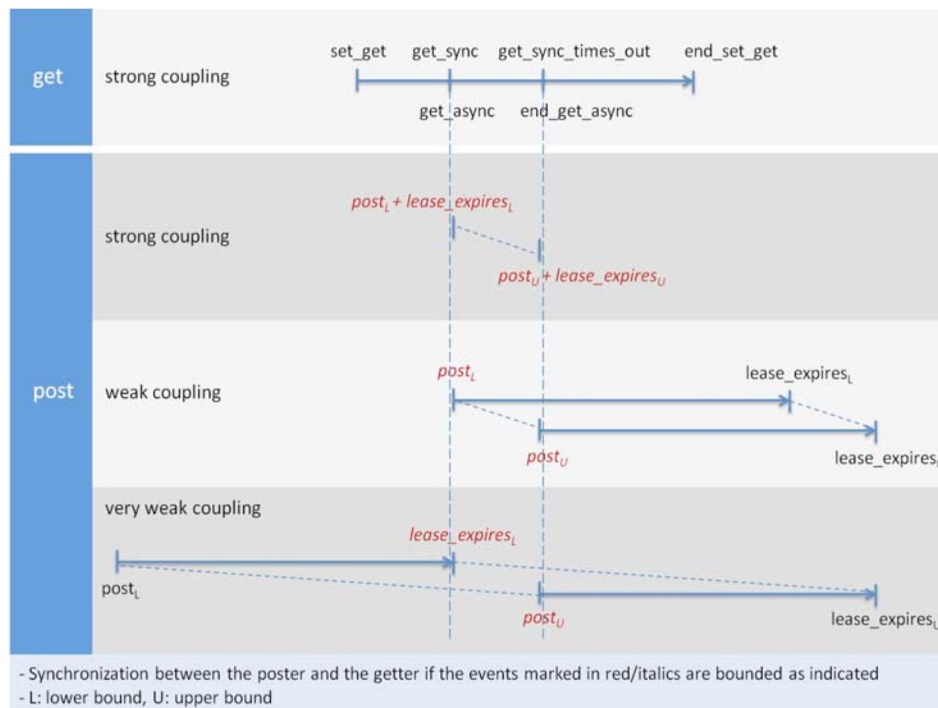


Figure 6: GA coordination semantics (glue with *get* under string coupling).

Based on our definition of coupling, coordination may occur only if the related actions of the getter and poster intersect in *space* and *time*. Overlap in space relates to the definition of the respective *scopes* by the getter and poster, which set the peer(s) that may get the posted data *via* the GA connector. As for overlap in time, it depends on the time at which the related actions are executed by the getter and poster, and the associated *timeout* and *lease* parameters. Figure **6** informally depicts the time synchronization constraints applying to the coordination semantics implemented by the GA glue according to the respective semantics of the getter and poster, where the getter applies strong coupling and the poster applies each one of the three couplings. Since it is not crucial for the purposes of the paper, we do not show here the other two cases for the getter's coupling. More precisely in the figure, a getter under strong coupling is connected after executing get_sync/get_async and before get_sync times out or end_get_async is executed (active interval). The getter synchronizes successfully in time with a poster under strong coupling if the post action is executed inside its active interval, since the data expire immediately. In the case of a poster under weak coupling, the data are maintained for lease time, only if the getter has already manifested its interest when the data are posted. However, this means again that, for successful synchronization, the getter should be inside its active interval when the data are posted. Finally, in the case of a poster under very weak coupling, the data are maintained for lease time anyway. This means that the post action may be executed also in advance of the getter's active interval (certainly not after), but in any case the posted data should not expire before the getter gets activated.

Based on the key concepts of CAS regarding architectural connectors that were introduced in the whole Section III, the coordination semantics of GA was formalized in [4]. Furthermore, in [23], we designed and developed an extended service bus that applies the GA abstraction to concretely enable interoperability among heterogeneous distributed systems. Finally, in [24], we modeled and analyzed some of the non-functional semantics of the base connectors and the GA connector. The contribution of this paper precisely lies in making clear our interoperability approach from the software architecture point of view.

4. CHOREOS COORDINATION PROTOCOLS: ABSTRACTING CHOREOGRAPHY BEHAVIOR

Accounting for the definitions of CHOReOS *component* and *connector*, we conclude the definition of CAS by adding the notion of CHOReOS *coordination protocol* that abstracts choreography behavior. Specifically, the CHOReOS *coordination protocol* introduces a higher, application-layer connector that defines system-wide behavior, based on the

connection of CHOReOS components through the middleware-layer connectors introduced in the previous section.

4.1. Choreography-Based Coordination in the FI

A choreography can be seen as a network of collaborating services, i.e., CHOReOS components, $S = \{S_1,\ldots,S_n\}$, that can simultaneously run. As described in Section III, components communicate/interact by means of the CHOReOS connectors, which implement various communication/interaction paradigms that are abstracted by the GA connector type for the sake of interoperability in the FI. In this setting, the notion of coordination protocol is crucial since, although a given set of services, if coordinated in the right way, can be used to achieve the specified choreography's goal and requirements, they can completely miss the goal or the requirements if coordinated in a different way. In order to deal with this problem, we use additional software entities and interpose them among the services participating to the specified choreography. The intent of these additional entities, hereafter referred to as *Coordination Delegates* (CDs), is to coordinate the interaction of the participant services (functionally and non-functionally abstracted as CHOReOS components) in a way that the resulting collaboration complies with the desired choreography. This is done by relying on the GA connector type, which connects the CHOReOS components and enables communication among them. Thus, CDs perform "pure coordination", at application layer, by exploiting an LTS-based behavioral specification of the participant services (Section II) and considering heterogeneous communication already mediated by means of the GA connector (middleware layer). Moreover, by relying on the service's functional and non-functional abstractions described in Section II, possible adaptation at the service interface level is already solved.

In light of several choreography problems discussed in the literature [25-29] (just to mention a few) related to issues such as *conformance check*, *realizability analysis*, and *realizability enforcement*, we consider the *choreography-based coordination problem* as a choreography synthesis problem that can be phrased as follows: *Given a choreography specification C and a set of CHOReOS components* $S = \{S_1, \ldots, S_n\}$, *derive (when possible) a set of coordination delegates* $CD = \{CD_1, \ldots, CD_n\}$ *that, after suitably assembled with the components in S, constitute a distributed realization of C.*

In the next section, we characterize the main "ingredients" required to tackle the choreography-based coordination problem from an architectural point of view. That is, we give an overall description of CAS by putting its constituent entities all together, i.e., CHOReOS components, CHOReOS connectors, and CHOReOS coordination delegates; and characterize the notion of FI choreography-based coordination. Differently from what is done in this paper, our previous work in [30-32] addresses the choreography-based coordination problem mainly focusing on issues concerning the adopted development process [32] and the implementation of algorithms needed for distributed coordination purposes [30,31], hence completely disregarding the software architecture perspective. This perspective is crucial since it serves as enabler for the work described in [30-32].

4.2. Abstractions for FI Choreography-Based Coordination

CAS is made of CHOReOS components where consumers and providers are composed with (e.g., wrapped by) their CDs. A CD is connected to other CDs, through CHOReOS connectors, in an asynchronous way. Furthermore, CDs can be connected to providers/consumers by CHOReOS connectors. Figure **7** illustrates an example of a CAS architecture.

Our choreography-based coordination style logically distinguishes between (i) *standard* and (ii) *additional communication.* The former denotes the operations that the components perform as described by their interface specification.

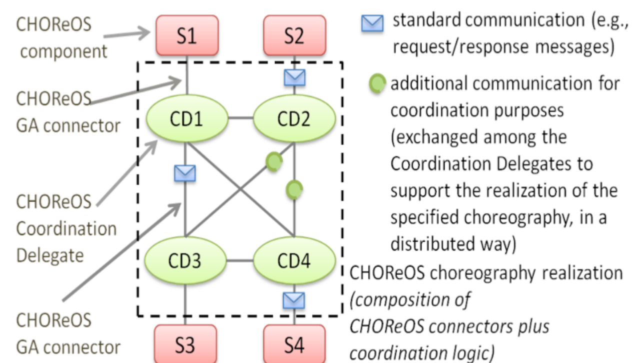


Figure 7: CAS architecture sample.

The latter denotes additional information that the CDs *asynchronously* exchange in order to coordinate each other. CDs prevent undesired interactions (violating the specified choreography) by coordinating

standard communication through the exchange of additional communication, when needed.

More specifically, sharing some similarities with [29], additional communication serves to keep track of the information about the "global state" of the coordination protocol (as implied by the specified choreography) that each CD can deduce from the observed standard communication flow. As better explained later, additional communication is exchanged only when synchronization is needed, i.e., when there is more than one component that is allowed to perform some action according to the current global state of the choreography model. Note that this limits the overhead due to the exchange of additional communication to the strictly necessary minimum.

We do not report the complete formal description of the abstractions that we use to characterize and reason on the entities of the CAS style. The full description can be found in [31,32]. In brief, we adopt an LTS-based specification to model the behavior of CHOReOS components observable from outside. Transitions are labeled with typed operation names so that *?a* (resp., *!a*) denotes a *provided* (resp., *required*) operation named *a* (see L_S in Figure **8a**).

We also use an LTS-based specification of the choreography, where the notions of role and operation abstract the notions of participant and task, respectively, of a BPMN2 choreography specification. Role names are used to label transitions in addition to operation names (see Figure **8b**).

A choreography LTS L_C can be then projected to one of its roles, denoted as $\pi_r(L_C)$, which is obtained by replacing every transition where *r* is neither the consumer nor the provider by a τ-transition. States linked by τ-transitions can be collapsed to a single state whose label denotes the set of collapsed states. Figure **8b** shows a Choreography LTS L_C, Figure **8c** shows its projection $\pi_{r1}(L_C)$, and Figure **8d** shows $\pi_{r1}(L_C)$ with collapsed states.

Indeed, as detailed in [30], within CHOReOS such LTS-based models are used to discover services whose behavior fulfills roles in the choreography. This is done by adopting a suitable notion of *refinement* based on *trace containment check*. For instance, the service model shown in Figure **8d** refines the projection $\pi_{r1}(L_C)$, i.e., the service can play *r1*.

The goal is to distribute L_C in a way that each CD knows which operation the supervised component is allowed to execute (*allowed operations*). Allowed operations are used by the CDs as a basis for correctly synchronizing with each other by exchanging additional communication. In other words, CDs interact with each other to restrict the components' standard communication by allowing only the part of the communication that is correct with respect to L_C.

According to the transitions of L_C, we record the allowed operations that can be required by the component C_i (playing the role r_i) from a global state *s* of L_C. We also record the other *active components* in *s*, i.e., the ones that can also require some (possibly different) allowed operation from *s*. Similarly, for each allowed operation we record the set of active components in the reaching state *s'*, i.e., the ones that can require or provide some allowed operation from *s'*. Thus, if a component C_i is going to perform an allowed operation from the state *s* of L_C, then all the other active components in *s* are *blocked* by sending a blocking message to the corresponding CDs; this is needed because otherwise, e.g., two different components could simultaneously proceed in two different states of L_C hence leading to a global state that is inconsistent with respect to L_C. Once C_i has performed the operation, all the components that can move in the new state *s'* of L_C are *unblocked* and they are made aware of the new current state *s'* of L_C; this is needed because some components that now can move could have been previously blocked. It is worth to note that, from the current state *s*, it is sufficient to block components only upon required operations. Components upon provided operations do not need to

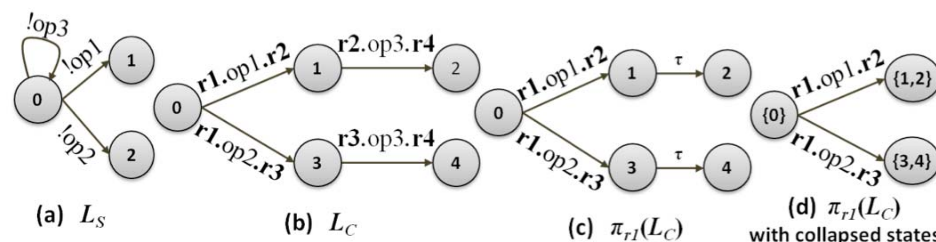


Figure 8: A behavior LTS model (**a**), a choreography LTS (**b**), its projection onto *r1* (**c**), with collapsed states (**d**).

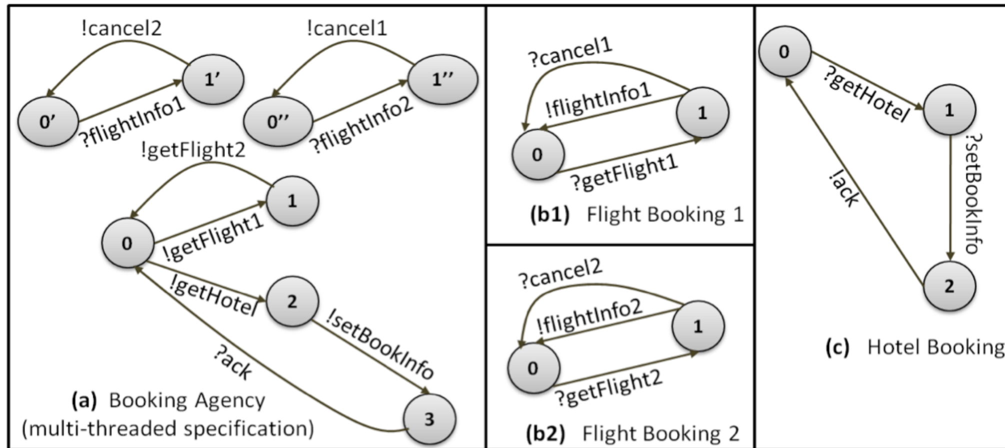


Figure 9: LTSs for travel agency services.

be blocked because they are already waiting for the operation request. Contrarily, once the new current state *s'* has been reached, components have to be unblocked upon both required and provided operations because, e.g., a component, that in *s'* provides an allowed operation, might have been blocked in *s* upon some other required operation.

Blocking and unblocking messages (together with acknowledging messages) concern the additional communication that is asynchronously exchanged among CDs for coordination purposes. It is worth to mention that the distribution of L_C into the various CDs, one for each component, can be efficiently obtained by means of a depth-first visit of L_C and, hence, its computational complexity is polynomial in the number of states of L_C. This computational complexity reveals that the elicitation of the distributed coordination logic out of the choreography LTS can be performed for large-scale contexts (with respect to the size of the choreography LTS).

As described in [31], within CHOReOS, we have implemented the distributed coordination algorithm that is performed by the CDs. The algorithm leverages on the *happened-before relation, partial ordering of events and time-stamp method* [33]. In this way an ordering among dependent blocking and unblocking messages is established and starvation problems are addressed. In [31], we fully formalize the algorithm, further proving its correctness and analyzing the negligible overhead due to the exchange of additional communication.

Illustrative example: Now, by means of a small example, we better show how CDs use, at run-time, additional communication according to the synthesized coordination information, to correctly and distributively interact with each other, hence ensuring the behavior globally specified by L_C. We consider the development of a choreography-based travel agency system that can be realized by choreographing four services: a Booking Agency service, two Flight Booking services, and a Hotel Booking service.

As shown in Figure **9,** starting from their initial states 0, Booking Agency makes requests to Flight Booking 1, Flight Booking 2, and Hotel Booking in order to book a flight and a hotel. The agency search for a flight by exploiting two different flight booking services. As soon as one of the two booking services answers by sending flight information (i.e., !flightInfo1 or !flightInfo2), the agency cancels the search on the other booking service (i.e., !cancel1 or !cancel2).

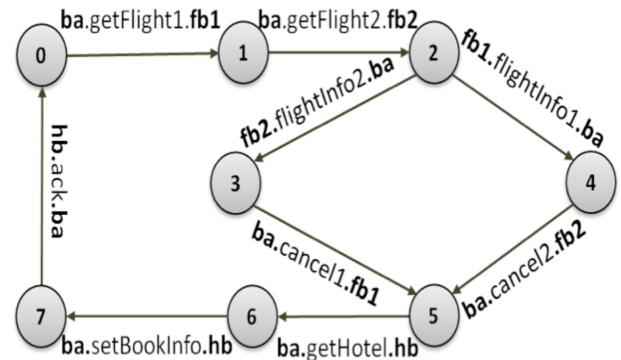


Figure 10: L_{FH}: a choreography LTS sample.

The choreography LTS L_{FH}, shown in Figure **10**, specifies that (i) flight booking has to be performed before hotel booking and (ii) only the answer from one of the two flight booking services is taken into account. Figure **11** shows how Flight Booking 2, playing the role fb2, is blocked whenever Flight Booking 1, playing the role fb1, is faster in collecting the information to be provided to Booking Agency, playing the role ba.

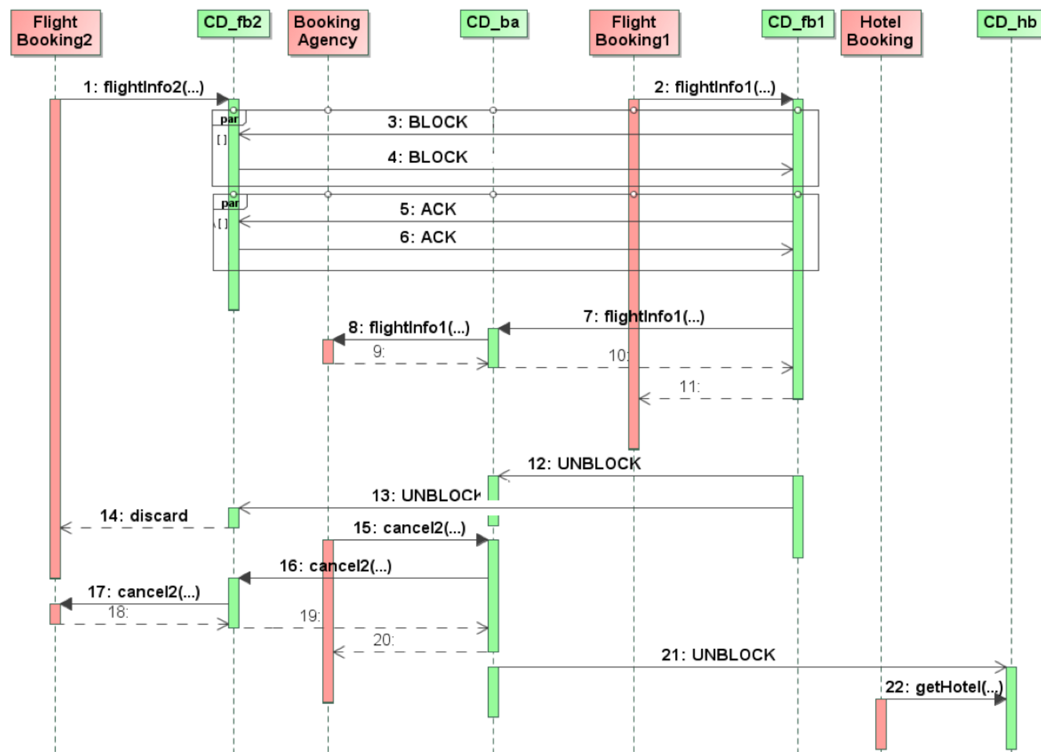


Figure 11: Synthesized choreography-based distributed coordination.

The shown scenario concerns an excerpt of the co-ordination enforced by the synthesized CDs. It starts when the two allowed operations flightInfo1 and flightInfo2, required by Flight Booking 1 and Flight Booking 2 respectively, concurrently occur while in the current state 2 of L_{FH}. At state 2, the timestamps for Flight Booking 1 and Flight Booking 2 are 1 and 2, respectively.

5. CONCLUSIONS AND FUTURE WORK

This paper reports a description of the architectural style defined within the CHOReOS EU project, namely the CHOReOS Architectural Style (CAS), to characterize choreography-based systems in the FI hence supporting their development and enactment. The main contributions of CAS can be summarized as follows:

- The CHOReOS service abstractions introduced in Section II enable sustaining the ultra large number of services aggregated in the FI, as well as their heterogeneity, in particular considering Business and Thing-based services that get inter-connected. The CHOReOS service abstraction specifically serves de- signing abstraction-oriented service bases for the support of service discovery and adaptation in the FI.
- The GA connector type that is defined in Section III enables revisiting the Enterprise Service Bus paradigm to enable cross-paradigm interaction and, in particular, interoperability across the Business and Thing domains.
- The formal abstractions for FI choreography-based coordination elaborated in Section IV paves the way for the automated synthesis of concrete distributed coordination protocols, from abstract choreography specifications, e.g., BPMN2 specification, and associated concrete services discovered in the FI.

As a result, CAS revisits the SOA paradigms to cope with the FI challenges, i.e., scalability, heterogeneity, mobility, and awareness & adaptability. CAS has been validated against three industrial use cases considered by CHOReOS, namely the "*Passenger-Friendly Airport*", the "*Mobile-enabled coordination of people*", and "*DynaRoute*" (see www.choreos.eu for details). Last but not least, the CAS specification informed the implementation of a supporting Integrated Development and Runtime Environment, *aka* the CHOReOS IDRE, whose components are released under open source license and promoted by the dedicated OW2 FISSi initiative on Future Internet Software Services. See https://www.ow2.org/view/Future_Internet for details.

As further work, we have identified some pending issues among which: accounting for continuous interactions, while this paper has concentrated on discrete ones; dealing with data-flow coordination, while this paper is mainly focused on control-flow coordination; and addressing choreography dynamic adaptation and evolution.

In this direction, several challenges, with respect to the current state of the art of CHOReOS, need to be addressed. Specifically, synthesis techniques are needed in order to infer an enhanced collaboration logic that, in addition to pure coordination, enables run-time choreography evolution in response of possible adaptations. The choreography synthesis problem is in general hard in the sense that not all possible collaborations can be effectively realized. Thus, the enhanced synthesis techniques we plan to develop in the future have to deal with a combination of specific choreography or integration patterns that correspond to service collaborations tractable *via* exogenous coordination. This approach also allows to produce parameterized coordination patterns off line that can adapted at run-time by providing dynamically sensed data. In particular, enhanced CDs will be interposed among the constituent services in order to enforce correct coordination logic with respect to the specified choreography and, by leveraging modularity, to enable dynamic adaptation and evolution according to possible changes. In this direction, we will make use of dedicated model transformations to generate, out of the choreography specification and the participant services interaction behavior, a model for each needed CD. The aim of an enhanced CD is twofold. On the one hand, it precisely describes the complex coordination logic implied by the choreography specification, and distributes it among the constituent services. On the other hand, by "projecting" the goal specification on the models of the service interaction behavior, it instantiates the inferred adaptation and evolution logic into a set of concrete adaptors, one for each constituent service, that dynamically and correctly filter service behavior in response of changes. All the above future issues will be the main subject of study of another European project called CHOReVOLUTION[1] that basically is a sequel of the CHOReOS project. This also means that, within CHOReVOLUTION, some members of the CHOReOS consortium will continue to collaborate on topics related to the automated synthesis of dynamic (aka adaptable and evolvable) choreographies for the FI.

[1]CHOReVOLUTION: Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet. Call: H2020-ICT-2014-1. Type of Action: RIA. Project no: 644178. Duration: 36 months. Start Date: 2015-01-01. Requested EU Contribution: €3,057,547.00.

ACKNOWLEDGEMENTS

This work was partly supported by the European Community's FP7/2007-2013 under Grant Agreement 257178 (project CHOReOS - www.choreos.eu).

REFERENCES

[1] Shaw M, Garlan D. Software architecture: perspectives on an emerging discipline. Prentice Hall 1996.

[2] Issarny V, Georgantas N, Hachem S, Zarras A, Vassiliadis P. Autili M, MA Gerosa A, Hamida B. Service-oriented middleware for the Future Internet: state of the art and research directions. J. Internet Services and Applications 2011; 2(1): 23-45. http://dx.doi.org/10.1007/s13174-011-0021-3

[3] Pautasso C, Zimmermann O, Leymann F. Restful Web Services vs. big Web Services: Making the Right Architectural Decision. In Proceedings of the 17th IEEE International Conference on World Wide Web 2008; 805-814. http://dx.doi.org/10.1145/1367497.1367606

[4] CHOReOS Project. Final CHOReOS Architectural Style and its Relation with the CHOReOS Development Process and IDRE - Deliverable D1.4b. September 2011.

[5] NEXOF-RA Project. The NEXOF-RA Ref. Model v3. www.nexofra.eu.

[6] Medvidovic N, Taylor R. A classification and comparison framework for software architecture description languages. TSE 2000; 26(1).

[7] Weber RH, Weber R. Internet of Things. Springer 2010.

[8] Guinard D, Trifa V, Karnouskos S, Spiess P, Savio D. Interacting with the SOA-based Internet of Things: Discovery query selection and on-demand provisioning of web services. TSC 2010; 3.

[9] Jammes F, Smit H. Service-oriented paradigms in industrial automation. TII 2005; 1(1).

[10] W3C. Web application description language. Tech Rep 2009. http://www.w3. org/Submission/wadl/.

[11] Athanasopoulos D, Zarras A, Vassiliadis P. Service Selection for Happy Users: Making User-Intuitive Quality Abstractions. In Proceedings of the 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) 2012; 32-36.

[12] CHOReOS Project. ntegrated CHOReOS Middleware - Enabling Large-Scale, QoS-Aware Adaptive Choreographies D3. 3. September 2013.

[13] CHOReOS Project. CHOReOS Dynamic Development Process: Methods and Tools- Deliverable D2.3. September 2013.

[14] Carzaniga A, Wolf A. Content-based Networking: A New Communication Infrastructure. LNCS 2002.

[15] Murphy AL, Picco GP, Roman GC. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. TOSEM 2006; 15(3).

[16] Issarny V, Bennaceur A, Bromberg YD. Middleware- layer connector synthesis: Beyond state of the art in middle- ware interoperability. in SFM 2011; 217-255.

[17] CHOReOS Project. CHOReOS State of the Art, Baseline and Beyond - Deliverable D1.1. December 2010.

[18] Papazoglou MP, Heuvel WJ. Service oriented architectures: approaches, technologies and research issues. The VLDB J 2007; 16.

[19] Avilés-López E, García-Macías J. TinySOA: a Service-oriented Architecture for Wireless Sensor Networks. Service Oriented Computing and Applications 2009; 3(2): 99-108. http://dx.doi.org/10.1007/s11761-009-0043-x

[20] Ceriotti M, Murphy AL, Picco GP. Data shar- ing vs. message passing: Synergy or incompatibility?: An implementation-driven case study. in SAC '08: Proceedings of the 2008 ACM symposium on Applied computing. New York NY USA: ACM 2008; 100-107.

[21] Blair G, Bennaceur A, Georgantas G, Grace P, Issarny V, Nundloll V, Paolucci M. The Role of Ontologies in Emergent Middleware: Supporting Interoperability in Complex Distributed Systems. in Middleware 2011.

[22] Eugster, Patrick Th, Felber, Pascal A, Guerraoui, Rachid, Kermarrec, Anne-Marie. The many faces of publish/subscribe. ACM Comput Surv 2003; 35(2): 114-131. http://dx.doi.org/10.1145/857076.857078

[23] Georgantas N, Bouloukakis G, Beauche S, Issarny V. Service-oriented Distributed Applications in the Future Internet: The Case for Interaction Paradigm Interoperability. In European Conference on Service-Oriented and Cloud Computing (ESOCC) 2013. http://dx.doi.org/10.1007/978-3-642-40651-5_11

[24] Kattepur A, Georgantas N, Issarny V. QoS Analysis in Heterogeneous Choreography Interactions, in 11th International Conference on Service Oriented Computing (ICSOC). Berlin, Germany. December 2013. http://dx.doi.org/10.1007/978-3-642-45005-1_3

[25] Marconi A, Pistore M, Traverso P. Automated composition of web services: the astro approach. IEEE Data Eng. Bull 2008; 31:(3).

[26] Melliti T, Poizat P, Mokhtar SB. Distributed be- havioural adaptation for the automatic composition of seman- tic services. in FASE 2008.

[27] Ben S, Mokhtar N, Georgantas, Issarny V. COCOA: Conversation-based service composition in pervasive computing environments with QoS support. JSS 2007; 80.

[28] Salaün G. Generation of service wrapper protocols from choreography specifications. In SEFM 2008.

[29] Basu S, Bultan T. Choreography conformance *via* synchronizability. in WWW, 2011.

[30] Autili M, Inverardi P, Tivoli M. Automated Synthesis of Service Choreographies. IEEE SOFTWARE. Special Issue on Software Engineering for Internet Computing: Internetware and Beyond 2014 (to appear).

[31] Autili M, Tivoli M. Distributed Enforcement of Service Choreographies, in: 13th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA'14).

[32] Autili M, Di Ruscio D, Di Salle A, Inverardi P, Tivoli M. A Model-Based Synthesis Process for Choreography Realizability Enforcement. In: Fundamental Approaches to Software Engineering (FASE'13). LECTURE NOTES IN COMPUTER SCIENCE 2013; 7793: 37-52: ISBN: 978-3-642-37056-4, ISSN: 0302-9743.

[33] Lamport L. Ti clocks, and the ordering of events in a distributed system. Commun. ACM 1978; 21.

Received on 05-10-2014 Accepted on 01-11-2014 Published on 18-11-2014

© 2014 Athanasopoulos *et al.*; Avanti Publishers.
This is an open access article licensed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted, non-commercial use, distribution and reproduction in any medium, provided the work is properly cited.