Automated Synthesis of Service Choreographies

Marco Autili, Paola Inverardi, and Massimo Tivoli, *University of L’Aquila, Italy*
{marco.autili, paola.inverardi, massimo.tivoli}@univaq.it

**Abstract**
The Future Internet promotes the production of a distributed computing environment that will be increasingly surrounded by a virtually infinite number of software services, which can be composed to meet user needs. Services will be increasingly active entities that, communicating in a peer-to-peer style, can proactively take decisions and autonomously engage in tasks. Service choreography is a form of decentralized service composition that describes peer-to-peer message exchanges occurring among participant services from a global perspective. In a distributed setting, obtaining the coordination logic required to realize a choreography is a non-trivial and error prone task. Automatic support to the realization of actual choreographies is then needed. To this purpose, we have developed a synthesis tool for the automatic realization of service choreographies.

**Keywords**
Future Internet, Service Choreographies, Automated Synthesis, Distributed Coordination.

The Future Internet (FI) era represents an age of unprecedented opportunities for social, economic, and business growth. FI promotes the existence of a distributed computing environment that will be increasingly surrounded by a virtually infinite number of software services. This enables the growth of innovative and revolutionary everyday-life scenarios within smart cities, and related software ecosystems, that ease human daily activities and give support to new markets and employment opportunities. A key enabler for this vision is the ability to automatically compose and dynamically coordinate software services.

Today’s service composition mechanisms are mostly based on service orchestration [2,8], which is a centralized approach to composing multiple services into a larger application. By analogy with music, orchestration can be seen as the process in which musicians play instruments following an orchestrator that represents a single point of control. Orchestration works well in rather static environments where services are predefined and environment changes are minimal. These assumptions are inadequate in the FI vision where a large number of diverse service providers and consumers keep changing and cannot be coordinated by using a centralized approach.

Differently, service choreography is a decentralized approach, which provides a looser way to design service composition by specifying participants and message protocols between them. By analogy with dance, choreography can be seen as a set of dancers that autonomously dance following a pre-established global scenario without a single point of control. The need for service choreography was recognized in BPMN2 ([www.omg.org/spec/BPMN/2.0](http://www.omg.org/spec/BPMN/2.0)), which introduces choreography modeling constructs. Specifically, service choreographies model peer-to-peer communication by defining a multi-party protocol that, when put in place by the cooperating parties, permits to reach the overall choreography objectives in a fully distributed way. In this sense, service choreographies are significantly different from service orchestrations in which a single stakeholder centrally plans and decides how an objective should be reached through cooperation with other services.
Future software systems cannot be realized by using orchestration only; they can be put in place solely if choreographies are also employed. Indeed, services will be increasingly active entities that, communicating in a peer-to-peer style, can proactively take decisions and autonomously engage in tasks according to both their own imminent needs and the emergent global collaboration.

REALIZING CHOREOGRAPHIES: THE PROBLEM
In choreography-based systems, the following two problems are usually considered: (i) realizability check – checks whether the choreography can be realized by implementing each participant so that it conforms to the choreography role that specifies its “expected” behaviour; and (ii) conformance check – checks whether the overall global interaction of a set of services satisfies the choreography. In the literature many approaches have been proposed to address these problems [1,3–10]. However, to put choreography in practice, we need to consider realizing service choreographies by reusing third-party services. This puts forward a further problem concerning automatic realizability enforcement. That is, given a choreography specification and a set of existing services (discovered or registered as possible participants), coordinate their interaction so to fulfill the global collaboration prescribed by the choreography specification.

REALIZING CHOREOGRAPHIES: OUR SOLUTION
Within the European Project CHOReOS (www.choreos.eu), we have developed a synthesis processor (CHOReOSyt − choreos.disim.univaq.it) for the realizability enforcement of service choreographies via automated generation of additional software entities, called Coordination Delegates (CDs). When interposed among the participant services, CDs proxify the services interactions and coordinate them in order to realize the specified choreography in a fully distributed way.

With reference to a scenario in the In-store Marketing and Sale domain, we will describe the choreography realization process that CHOReOSyt implements. We further point out how we advance the state of the art towards automating the choreography realizability enforcement process. In particular, the main novelties of CHOReOSyt concern the ability to: (i) automatically produce CDs without passing through the generation of a centralized model of the whole system, hence avoiding the state explosion problem; (ii) support the whole CDs synthesis process, from the choreography specification to the CDs actual code generation, hence going beyond producing abstract models of the CDs only; and (iii) take into account all the complex coordination constructs of the BPMN2 notation, which is the de facto standard adopted in today’s choreography specification practice.

REFERENCE SCENARIO
Within CHOReOS, CHOReOSyt has been experimented on a reference scenario of daily life that requires the complex coordination of business services, thing-based services, and stakeholders belonging to three different domains: passenger-friendly air transportation, customer relationship management, and intelligent transportation. This scenario involves a number of different, yet correlated, choreographies. Figure 1 shows the BPMN2 Choreography Diagram drawn with Eclipse BPMN2 Modeler (eclipse.org/bpmn2-modeler). It concerns the In-store Marketing and Sale (sub-)choreography of a customer relationship management system, which aims to monitor the activity of a client inside a shop in order to propose customized shopping offers and advertisements. By exploiting a store totem service, the
marketing application service cooperates with the marketing manager service to manage also the offers of public products inside the store.

Figure 1. In-store Marketing and Sale BPMN2 choreography. In BPMN2, a choreography task is an atomic activity that represents an interaction by means of one or two message exchanges (request and optionally response) between two participants. Graphically, BPMN2 choreography diagrams use rounded-corner boxes to denote choreography tasks. Each task is labeled with the roles of the two participants involved in the task (e.g., see Client and Shop Entrance), and the name of the task (e.g., see Assign a Cart after Check-in) performed by the initiating participant and provided by the other one. A role contained in the white box denotes the initiating participant (Client). Note that task and message descriptions are specified by using XML schema.

The choreography is triggered by the Client entering the shop. The Shop Entrance service detects the presence of a specific Client inside the store, assigning her a cart and notifying the Marketing Application about the new Client. Once the Client has subscribed to her virtual cart, she can perform shopping by adding and removing products from it. While Clients are inside the shop, both customized and public marketing activities are performed. The Marketing Application keeps asking the Marketing Manager for new public offers and advertisements, which are then pushed by the Marketing Application itself onto the
In-store Totems. Private offers and advertisements are also constantly requested by the Marketing Application to the Marketing Manager, and are delivered to the Clients inside the shop by the SMS service and by sending them to their Shopping Assistant Apps. Once the Client finishes her shopping activities, she goes to a Self Check-out Machine to perform payment. This is achieved with the interaction between the Self Check-Out and the Smart Cart services.

This choreography uses many parallel (see the parallel and merging branches represented as rhombus marked with “+” with two outgoing and two incoming arrows, respectively) and alternative flows (see the alternative exclusive branches represented as rhombus marked with “x” with two outgoing arrows) that give rise to many concurrent execution flows that must be properly coordinated. In fact, it contains a number of forking branches, nested with conditional branches, that must be joined at later execution points. Moreover, the choreography runs in a distributed setting in which the participant services are active entities executing concurrently, and may not synchronize as prescribed by the choreography. As a result, the global collaboration might exhibit undesired interactions. That is, interactions that are not allowed by the choreography specification, but can happen when the services collaborate in an uncontrolled way. For instance, a Client is allowed to perform the Add Product task in order to add products to the Smart Cart (see the top side of the choreography diagram). However, after the Payment has been accomplished and before Checking out, a “malicious” client may attempt at adding further products (see the top-most tasks just before the End event), hence avoiding to pay for them.

The scenario described above reveals that implementing the required coordination logic is a non-trivial and error prone task in a distributed setting. Automatic support to the realization of “correct” actual choreographies is therefore desirable.

CHOREOGRAPHY REALIZATION PROCESS

By referring to the discussed scenario, Figure 2 shows the process that we have realized to automatically synthesize the involved choreographies.

Figure 2. Process overview
**Step 1** – Software producers cooperate with domain experts and business managers in order to set the business goal (e.g., assist travellers from arrival, to staying, to departure), identify the tasks and the participants required to achieve the goal (e.g., reserving taxi to the local taxi company, purchasing digital tickets into the train station, as well as performing transactions by means of NFC-based services located into a shop), and specify the way the participants have to collaborate through a BPMN2 choreography diagram. CHOREOS supports this step by providing a plugin that allows to import the goal specification into the MagicDraw modeling tool (www.nomagic.com) and associate it with BPMN2 constructs along with QoS constraints. In particular, CHOREOS uses the Q4BPMN notation – an extension to BPMN2 – for specifying non-functional properties and dedicated automated tools for performing quality assessment of the choreography specification.

**Step 2** – The modeled choreography is exported by MagicDraw and given as input to CHOREOSynt. Indeed, CHOREOSynt supports the XML-based encoding of BPMN2 choreographies, such as the one of the Eclipse BPMN2 Modeler.

**Step 3** – CHOREOSynt queries the registry to discover services that are suitable to play the roles of the choreography. The registry contains services published by providers (e.g., transportation companies and airport retailers) that have identified business opportunities in the domain of interest. WSDL (www.w3.org/TR/wsdl) is used for describing service interfaces. BPEL (www.oasis-open.org) is used for describing services interaction behavior by specifying the flow of messages that are exchanged with the environment. The registry also contains the registration of the end users interested in exploiting the choreography through their mobile apps.

**Step 4** – Starting from the BPMN2 choreography diagram and the set of discovered services, CHOREOSynt synthesizes a set of CDs. The synthesis step exploits model transformations. The transformations have been implemented by means of the Atlas Transformation Language (ATL – www.eclipse.org/atl), which is a domain specific language for realizing M2M transformations. Essentially, the developed ATL transformations consist of a number of rules each devoted to the management of specific BPMN2 modeling constructs. The current implementation of these transformations within CHOREOSynt (available at choreos.disim.univaq.it) extends and advances their preliminary version in [11,12].

**Step 5** – The generated CDs together with the description of the services are given as input to the Enactment Engine for deployment. CD deployment descriptors are codified in XML.

**Step 6** – CDs are interposed among those participant services that need to be coordinated. CDs perform coordination of the services’ interaction in a way that the resulting collaboration realizes the specified choreography. To achieve correct coordination, CDs exchange at run-time additional communication to prevent undesired interactions. CD additional communication is codified in XML. The coordination logic embedded in CDs is obtained by a distributed coordination algorithm implemented in Java; each CD runs its own instance of the algorithm. Once deployed by the Enactment Engine component, CDs support the correct execution of the
choreography by realizing the required distributed coordination logic among the discovered services.

**CHOREOGRAPHY SYNTHESIS PROCESSOR**

As shown in Figure 3, the architecture of CHOREOSynt comprises four main REST services that perform the synthesis-time activities shown in Figure 2. The figure also reports the main functionalities offered by CHOREOSynt, whose detailed description is given below.

![Figure 3. Architecture of the Choreography Synthesis Processor](image)

**M2M Transformer** – The Model-to-Model (M2M) Transformer offers a REST operation `bpmn2clts()` that implements an ATL-based transformation. It takes as input the BPMN2 specification of the choreography and automatically generates an automata-based model, called Choreography LTS (CLTS). A CLTS is a finite state automaton that, for coordination purposes, is suitably extended with fork and join constructs, conditional branching and conditional loops (i.e., typical constructs of BPMN2). The CLTS provides CHOREOSynt with a formal choreography model that is independent from the specific choreography modeling notation. This means that CHOREOSynt can be used in different practical contexts where different modeling notations might be adopted, provided that a dedicated M2M transformation is implemented.

Starting from the CLTS, the synthesis process extracts the list of the participants and automatically derives the CLTSs modeling the expected behavior for each of them. To this end, another operation named `extractParticipants()` is offered. The participant-specific CLTS is produced by projecting the choreography onto the role played by the participant, hence filtering out the transitions, and related states, that do not belong to the role description. The participant-specific CLTS models the interaction behavior
that a candidate service (to be discovered out of the registry) has to support in order to be able to play the role of the participant in the choreography.

**Synthesis Discovery Manager** – As also shown in Figure 2, the synthesis process and the discovery process interact each other to retrieve, from the registry, candidate services that are suitable for playing the participant roles required by the choreography, that is those services whose behaviors are compatible with the participant-specific CLTS. In particular, for each participant, the Synthesis Discovery Manager component interacts with the eXtensible Service Discovery (XSD) system by invoking the `discoverServices()` operation that takes the participant-specific CLTS as input.

**Behavior Simulator** – After candidate services have been discovered, the synthesis process selects the ones that can behaviorally simulate a participant-specific CLTS. To this purpose, the Behavior Simulator offers an operation named `simulation()`. It takes as input the participant-specific CLTS and the BPEL description of the service retrieved by the XSD service. It is worth mentioning that the simulation method implements a notion of LTS simulation suitably extended to treat CLTSs and BPEL. That is, a BPEL description simulates a participant-specific CLTS if the former specifies – at least – all the message flows modeled by the latter. In other words, this means that the behavior of the discovered service must cover the expected-behavior of the participant.

**Coordination Delegate Generator** – After the services have been selected for all the choreography participants, the synthesis processor can generate the CDs through the operation `generateCD()` offered by the Coordination Delegate Generator component. This component produces an executable description of the choreography to be passed to the Enactment Engine through the operation `createChorSpec()`. It takes as input the selected services and the CDs generated for them. In particular, the choreography executable description is an XML-based declarative description that specifies the locations of the selected services, the locations of the generated CDs, and the service-CD and CD-CD interdependencies.

**CHOReOSynt ECLIPSE PLUGIN**
CHOReOSynt has been implemented in two modalities. The user may choose whether to execute the *Automatic Synthesis Processor* or the *Interactive Synthesis Processor*. The former produces all the artefacts in one step; the latter produces the artefacts step by step, and visualizes them by using a GMF-based (www.eclipse.org/modeling/gmp) graphical editor we have developed. The synthesis plugin can be automatically installed into the Eclipse platform through the update site: [http://choreos.disim.univaq.it/updatesite/site.xml](http://choreos.disim.univaq.it/updatesite/site.xml). A screencast video demonstrating CHOReOSynt at work on the reference scenario is available at [choreos.disim.univaq.it](http://choreos.disim.univaq.it).

**STATE OF THE ART AND BEYOND**
The CHOReOSynt approach is related to a number of other approaches that have been considered in the literature in the domain of service-oriented engineering. For space reasons, we discuss only the approaches closer to the automated choreography enforcement problem.

In [5] the authors propose an approach to enforce the realizability of a given choreography via automated generation of monitors. Each monitor acts as a local controller for its peer. Monitors are obtained through the iteration of equivalence checking steps between two centralized models of the whole system. One
model (resp., the other) is produced by composing the peer LTSs assuming synchronous (resp., asynchronous) communication. The notion of monitor discussed in [5] is similar to the notion of CD. However, our approach synthesizes CDs without producing a centralized model of the whole system, hence preventing the state explosion problem. Furthermore, the approach in [5] is more on the theoretical side and focuses on generating the model of the monitors only; whereas, we focus on synthesizing both the actual code implementing the CDs and their deployment schema.

In [6] the authors present an analysis approach to check the conformance between the choreography specification and the composition of participant implementations. Their framework is capable of modeling and analyzing compositions, where the interactions can be also asynchronous, and the messages can be stored in unbounded queues and reordered if needed. Following this line of research, in [7], the authors provide a hierarchy of realizability notions that forms the basis for a more flexible analysis with respect to classical realizability checks. Although the approaches described in [6,7] are novel in the sense that they characterize relevant properties to check a certain realizability degree, they remain in the domain of statically checking realizability and do not deal with its automated enforcement at run time.

In [8] the ASTRO toolset is demonstrated. ASTRO offers functionalities to support the automated composition of web services and the monitoring of their execution. The goal of the automated composition phase is to achieve a new composed service starting from a business requirement and the description of the protocols defining available external services. More specifically, a planner component synthesizes automatically the code of a centralized process that achieves the business requirement by interacting with the available external services. Differently from our approach ASTRO deals with centralized orchestration-based business processes, rather than fully decentralized choreography-based ones.

As described in [9], the CIGAR framework aims at solving the (multi-)goal recognition problem. CIGAR is essentially able to decompose an observed sequence of multi-goal activities into a set of action sequences, one for each goal, specifying whether a goal is active or not in a specific action. Although this kind of goal decomposition to some extent recalls the choreography decentralization that CHOReOSynt performs, goal recognition represents a fundamentally different problem with respect to realizability enforcement. That is, goal recognition concerns learning a goal-based model of an agent from an observation of the actions it takes while interacting with the environment. Differently, realizability enforcement concerns the production of decentralized coordination logic out of a task-based specification of the choreography.

The work in [10] presents the TCP-Compose algorithm that, given a set of candidate services offering the desired functionalities, identifies the set of composite services that better fit the user-specified qualitative preferences over non-functional attributes. As far as the CHOReOSynt process is concerned, the work in [10] might be exploited to realize an extension of the discovery process so to enable a more flexible selection of services from the registry.

The work described in this paper represents an advance also with respect to our previous work in [11,12]. In fact, although the synthesis process described there treats most of the BPMN2 constructs, it considers a simplified version of their actual semantics. For instance, as also done in [5], the selection of conditional
branches is simply abstracted as a non-deterministic choice, regardless the run-time evaluation of their enabling conditions. Analogously, parallel flows are enforced by non-deterministically choosing one of their linearizations obtained through interleaving, hence loosing the actual parallelism degree. To overcome these limitations, CHOReOSynt relies on a choreography model that, being more expressive than the choreography model adopted in [9,10], preserves the actual semantics of the BPMN2 constructs. Relying on a more expressive model has permitted us to define a novel and more effective distributed coordination algorithm.

CONCLUDING DISCUSSION
The experiments carried on within CHOReOS demonstrated that CHOReOSynt can be effectively applied in practical contexts. In particular, our experiments show that considering domain-specific interaction patterns mitigates the complexity of the coordination enforceability problem when recurrent business protocols have to be enforced. Indeed, in general, the choreography synthesis problem is hard in the sense that not all possible collaborations can be automatically realized. This suggests that CHOReOSynt may be better exploited with a combination of domain-specific choreography patterns, as well as protocol interaction patterns that correspond to service collaborations tractable via exogenous coordination. This approach would also allow to produce parametrized coordination patterns offline, which can be then instantiated at run-time. Currently, CHOReOSynt supports pure coordination. It does not deal with heterogenous interaction protocol adaptation since mismatches at the level of service operations and related I/O parameter types are not accounted for. In order to support data-based coordination through the elicitation and application of complex data mappings, CHOReOSynt should be enhanced in order to automatically infer mappings to match the data types of messages sent or received by mismatching participant services. This means effectively coping with heterogeneous service interfaces and dealing with as much Enterprise Integration Patterns [13] and Protocol Mediation Patterns [14] as possible, in a fully automatic way. In this direction, we achieved promising results in previous work on automated synthesis of modular mediators [15].

In order to enable the market take-up and further enhancement of the CHOReOSynt implementation by third-party developers, especially SMEs, including development of new applications to be commercialized, we released CHOReOSynt under the umbrella of the Future Internet Software and Services initiative (FISSI – www.ow2.org/view/Future Internet). FISSi is an activity set to develop awareness for OW2 Future Internet software towards members and non-members alike, open source vendors and proprietary vendors alike, with a market oriented approach. Our primary objective, to be achieved in the near future, is to establish a community of developers and third-party market stakeholders (e.g., users, application vendors, policy makers, etc.) around CHOReOSynt.

Acknowledgements
This work is supported by the Ministry of Education, Universities and Research, prot. 2012E47TM2 (project IDEAS – Integrated Design and Evolution of Adaptive Systems), and by the European Community FP7/2007-2013 under grant agreement number 257178 (project CHOReOS – Large Scale Choreographies for the Future Internet).
References

About the Authors

Marco Autili is Assistant Professor at the University of L’Aquila, Italy. His research focuses on automated synthesis for composing distributed systems, formal specification and checking of temporal properties, context-oriented programming and resource-oriented analysis of adaptable (mobile) applications. He received a PhD in Computer Science from University of L’Aquila. Mailing address: Marco Autili – Dip. di Ingnereria e Science dell’Informazione e Matematica (DISIM) – Università degli Studi dell’Aquila – Via Vetoio s.n.c. – I-67100 L’Aquila, Italy.
Phone: +39 0862 43 3186 – Fax: +39 0862 43 3131.
Paola Inverardi is Full Professor at the University of L’Aquila, Italy. Her research focuses on software specification and verification of concurrent and distributed systems, deduction systems, and Software Architectures. Mailing address: Paola Inverardi – Dip. di Ingegneria e Scienze dell’Informazione e Matematica (DISIM) – Università degli Studi dell’Aquila – Via Vetoio s.n.c. – I-67100 L’Aquila, Italy.
E-mail: paola.inverardi@univaq.it.

Massimo Tivoli is Assistant Professor at the University of L’Aquila, Italy. His research focuses on component adaptation and coordination, connector synthesis, software models elicitation, and choreography synthesis. He received a PhD in Computer Science from University of L’Aquila. Mailing address: Massimo Tivoli – Dip. di Ingegneria e Scienze dell’Informazione e Matematica (DISIM) – Università degli Studi dell’Aquila – Via Vetoio s.n.c. – I-67100 L’Aquila, Italy.
E-mail: massimo.tivoli@univaq.it.