Automated Synthesis of Service Choreographies

Article *in* IEEE Software · January 2015

DOI: 10.1109/MS.2014.131

CITATIONS
35

READS
582

3 authors:

Marco Autili
Università degli Studi dell'Aquila
118 PUBLICATIONS **1,538** CITATIONS
SEE PROFILE

Paola Inverardi
Università degli Studi dell'Aquila
333 PUBLICATIONS **9,139** CITATIONS
SEE PROFILE

Massimo Tivoli
Università degli Studi dell'Aquila
127 PUBLICATIONS **3,221** CITATIONS
SEE PROFILE

Automated Synthesis of Service Choreographies

Marco Autili, Paola Inverardi, and Massimo Tivoli,
University of L'Aquila

// Service choreographies describe peer-to-peer message exchanges among participant services from a global perspective. A proposed tool automatically synthesizes choreographies and is effective in practical contexts. //

FUTURE INTERNET RESEARCH promotes a distributed-computing environment that will be increasingly inhabited by a virtually infinite number of software services. This will enable the growth of innovative, revolutionary everyday scenarios in smart cities, and related software ecosystems, that ease human activities and support 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 based mostly on *service orchestration*, a centralized approach to composing multiple services into a larger application.[1,2] Orchestration works well in rather static environments with predefined services and minimal environment changes. These assumptions are inadequate in the Future Internet vision, in which many diverse service providers and consumers keep changing and can't be coordinated through a centralized approach.

In contrast, *service choreography* is a decentralized approach that provides a looser way to design service composition by specifying participants and message protocols between them. The need for service choreography was recognized in BPMN2 (Business Process Model and Notation Version 2.0; www.omg.org/spec/BPMN/2.0), which introduced choreography-modeling constructs. Service choreographies model peer-to-peer communication by defining a multiparty protocol that, when put in place by the cooperating parties, allows reaching the overall choreography objectives in a fully distributed way. In this sense, service choreographies differ significantly from service orchestrations, in which one stakeholder centrally determines how to reach an objective through cooperation with other services.

Future software systems won't be realized by orchestration only; they'll also require choreographies. Indeed, services will be increasingly active entities that, communicating peer-to-peer, proactively make decisions and autonomously perform tasks according to their own imminent needs and the emergent global collaboration.

In a distributed setting, obtaining the coordination logic required to realize a choreography is nontrivial and error prone. So, automatic support for realizing choreographies is needed. For this purpose, we developed CHOReOSynt (choreos.disim.univaq.it), a choreography synthesis tool. As you'll see, CHOReOSynt advances the state of the art in automating choreography realizability enforcement.

Realizing Choreographies: The Problems

Choreography-based systems usually consider two problems:

0740-7459/15/$31.00 © 2015 IEEE

- A *realizability check* determines whether the choreography can be realized by implementing each participant so that it conforms to the choreography role specifying its expected behavior.
- A *conformance check* determines whether the overall global interaction of a set of services satisfies the choreography.

To address these problems, researchers have proposed many approaches[2–5] (for a look at some approaches, see the sidebar). However, to put choreography into practice, we must consider realizing service choreographies by reusing third-party services. This leads to a further problem concerning automatic realizability enforcement. That is, given a choreography specification and a set of services (discovered or registered as possible participants), we must coordinate their interaction so as to fulfill the global collaboration that the choreography specification prescribed.

Our Solution

We developed CHOReOSynt as part of the European project CHOReOS (Large-Scale Choreographies for the Future Internet; www.choreos.eu). CHOReOSynt automatically generates software entities called *coordination delegates* (CDs). When interposed among the participant services, CDs "proxify" and coordinate the service interactions to realize the specified choreography in a fully distributed way.

CHOReOSynt has three unique abilities. First, it automatically produces CDs without generating a centralized model of the whole system, thus avoiding state explosion. Second, it supports the whole CD synthesis process, from choreography specification to CD code generation, thus going beyond producing only abstract models of the CDs. Finally, it takes into account all the complex coordination constructs of the BPMN2 notation, the de facto standard for choreography specification.

> Service choreography is a decentralized approach that provides a looser way to design service composition.

The Reference Scenario

We experimented with CHOReOSynt using a real-world reference scenario that requires the complex coordination of business services, thing-based services, and stakeholders from the customer relationship management (CRM) domain.

Figure 1 shows a BPMN2 choreography diagram drawn with the Eclipse BPMN2 Modeler (eclipse.org/bpmn2-modeler). It concerns the in-store marketing and sales subchoreography of a CRM system that monitors the activity of a customer (the Client in Figure 1) in a store to propose customized shopping offers and advertisements. By exploiting the In-Store Advertisement Totem service, the Marketing Application service cooperates with the Marketing Manager service to manage the offers of products in the store.

The choreography is triggered when the Client enters the store. The Shop Entrance service detects the Client's presence, assigning her a cart and notifying the Marketing Application about the new Client. Once the Client has subscribed to her virtual cart, she can shop by adding products to it or removing them. While the Client is in the shop, both customized and public marketing activities occur. The Marketing Application keeps asking the Marketing Manager for new public offers and advertisements, which the Marketing Application then pushes onto the In-Store Totem. The Marketing Application also constantly requests private offers and advertisements from the Marketing Manager. The SMS (short message service) service delivers the offers and advertisements to the Client by sending them to the Client's Shopping Assistant app. Once the Client finishes shopping, she goes to the Self Check-Out Machine to pay. This involves interaction between the Smart Cart and Self Check-Out services.

This choreography uses many parallel and alternative flows that give rise to many concurrent execution flows requiring proper coordination. (In Figure 1, the parallel and merging branches are rhombuses marked with "+" with two outgoing and two incoming arrows, respectively. The alternative exclusive branches are rhombuses marked with "×" with two outgoing arrows.) In fact, the choreography contains a number of forking branches, nested with conditional branches, that must be joined at later execution points. Moreover, it runs in a distributed setting in which the participant services are active entities executing

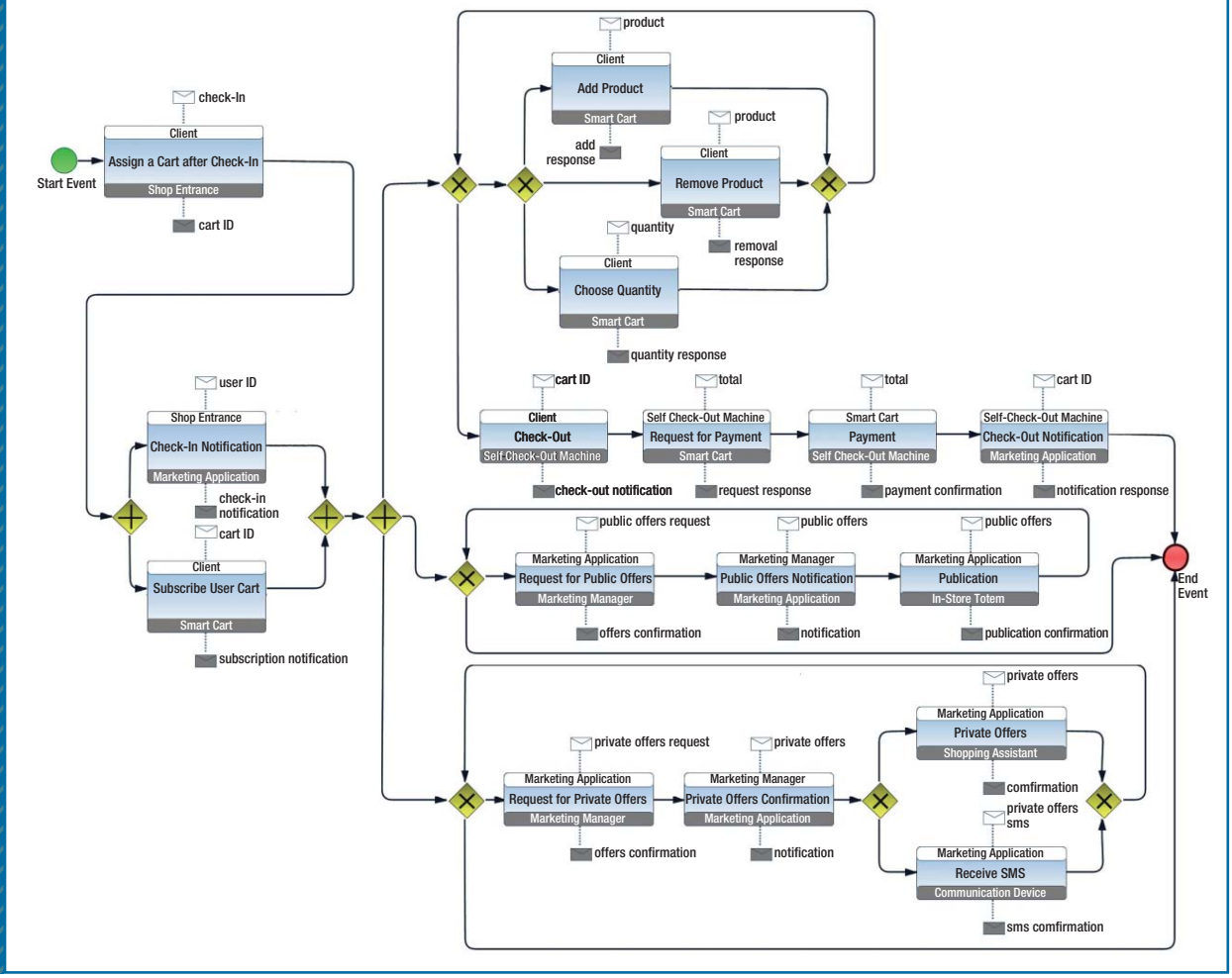

FIGURE 1. An in-store marketing and sales BPMN2 (Business Process Model and Notation Version 2.0) choreography. In BPMN2, a choreography task is an atomic activity representing an interaction through one or two message exchanges (a request and an optional response) between two participants. Graphically, BPMN2 choreography diagrams use rounded-corner boxes to denote choreography tasks. Each task is labeled with the two participants' roles (for example, Client and Shop Entrance), and the name of the task (for example, Assign a Cart after Check-In) performed by the initiating participant and provided by the other participant. A role in a white box denotes the initiating participant. Task and message descriptions are specified using XML schema.

concurrently and might not synchronize as the choreography prescribes.

So, the global collaboration might exhibit undesired interactions. 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 top of the Figure 1). However, after paying and before check-out, the Client might try to add products (see the top-most tasks just before the End Event), thus avoiding paying for them.

This scenario reveals that implementing the required coordination logic is nontrivial and error prone in a distributed setting. So, automatic support for realizing correct choreographies is desirable.

Choreography Realization

Using the previous scenario, Figure 2 diagrams our automatic choreography synthesis.

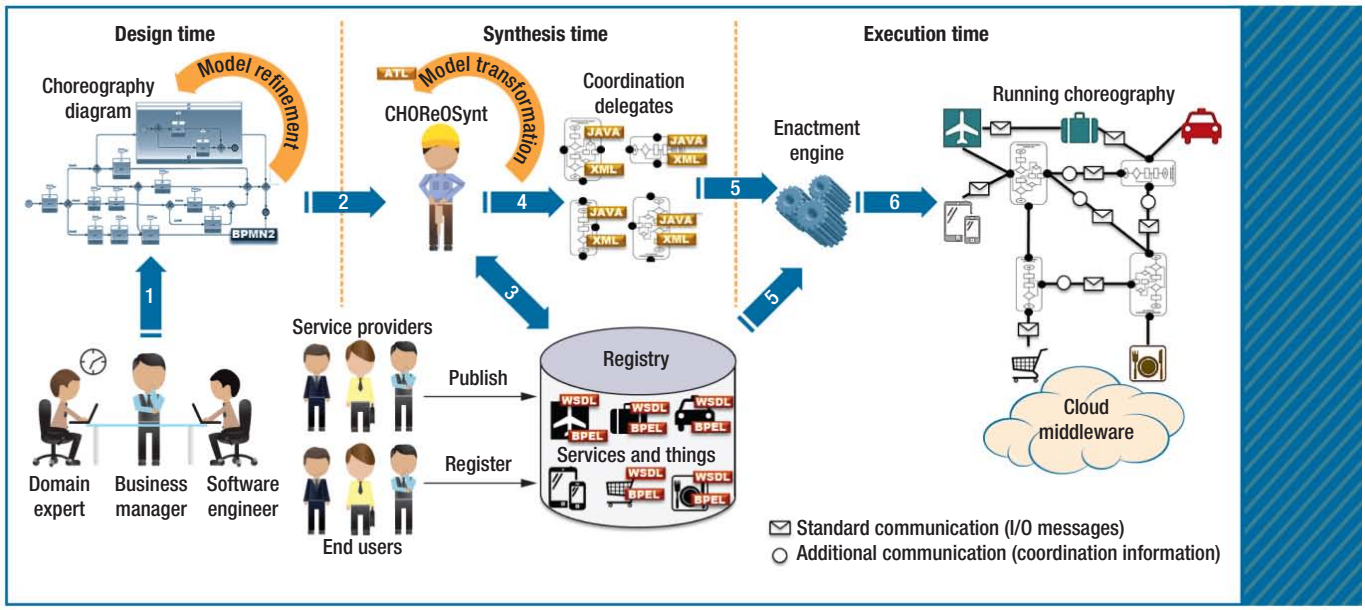

FIGURE 2. An overview of automatic choreography synthesis, using a scenario involving the coordination of business services, thing-based services, and stakeholders from air transportation, customer relationship management, and intelligent transportation. WSDL stands for Web Services Description Language; BPEL stands for Business Process Execution Language.

Step 1. Software producers cooperate with domain experts and business managers to

- set the business goal (for example, assist travellers from arrival, to staying, to departure),
- identify the tasks and participants required to achieve the goal (for example, reserving a taxi from the local taxi company, purchasing digital tickets at the train station, and performing transactions through services based on near-field communication in a shop), and
- specify how participants must collaborate through a BPMN2 choreography diagram.

To support this step, CHOReOS provides a plug-in that allows importing the goal specification into the MagicDraw modeling tool (www.nomagic.com) and associates it with BPMN2 constructs and quality-of-service constraints. In particular, CHOReOS uses both the Q4BPMN notation—an extension to BPMN2—to specify nonfunctional properties and dedicated automated tools to assess the choreography specification's quality.

Step 2. MagicDraw exports the modeled choreography to CHOReOSynt. CHOReOSynt supports the XML-based encoding of BPMN2 choreographies, such as the one of the BPMN2 Modeler.

Step 3. CHOReOSynt queries the registry to discover services suitable for playing the choreography's roles. The registry contains services published by providers (for example, transportation companies and airport retailers) that have identified business opportunities in the domain of interest. To describe service interfaces, CHOReOSynt uses WSDL (Web Services Description Language; www.w3.org/TR/wsdl). To describe service interaction behavior, BPEL (Business Process Execution Language) specifies the flow of messages exchanged with the environment. The registry also contains the registration of users interested in exploiting the choreography through their mobile apps.

Step 4. Starting from the choreography diagram and the set of discovered services, CHOReOSynt synthesizes a set of CDs. The synthesis exploits model transformations. The transformations are implemented through ATL (www.eclipse.org/atl), a domain-specific language for realizing model-to-model (M2M) transformations. ATL transformations comprise a number of rules, each of which manages a specific BPMN2 modeling construct. The current implementation of these transformations in CHOReOSynt (available at

THE STATE OF THE ART AND BEYOND

CHOReOSynt (see the main article) is related to several approaches developed for service-oriented engineering. For space reasons, we discuss only the approaches closest to automated choreography enforcement.

Matthias Güdemann and his colleagues' approach enforces a choreography's realizability by automatically generating monitors.[1] 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. It produces one of the models by composing the peer *labeled transition systems* (LTSs) assuming synchronous communication. It produces the other by composing the peer LTSs assuming asynchronous communication.

Güdemann and his colleagues' monitor is similar to our coordination delegate (CD). However, our approach synthesizes CDs without producing a centralized model of the whole system, hence preventing state explosion. Furthermore, Güdemann and his colleagues' approach is more theoretical and generates only the model of the monitors. In contrast, our approach synthesizes both the actual code implementing the CDs and their deployment schema.

Raman Kazhamiakin and Marco Pistore's approach checks the conformance between the choreography specification and the composition of participant implementations.[2] Their framework can model and analyze compositions in which the interactions can also be asynchronous and the messages can be stored in unbounded queues and reordered if needed. Following this line of research, Kazhamiakin and Pistore provided a hierarchy of realizability notions that forms the basis for a more flexible analysis regarding classic realizability checks.[2,3] These two approaches are novel in that they characterize relevant properties to check a certain degree of realizability. However, they statically check realizability and don't automatically enforce it at run time.

The ASTRO toolset supports automated composition of Web services and the monitoring of their execution.[4] It aims to compose a service starting from a business requirement and the description of the protocols defining available external services. More specifically, a planner component automatically synthesizes the code of a centralized process that achieves the business requirement by interacting with the available external services. Unlike our approach, ASTRO deals with centralized orchestration-based business processes rather than fully decentralized choreography-based ones.

The CIGAR (Concurrent and Interleaving Goal and Activity Recognition) framework aims for multigoal recognition.[5] CIGAR decomposes an observed sequence of multigoal activities into a set of action sequences, one for each goal, specifying whether a goal is active in a specific action. Although such goal decomposition somewhat recalls CHOReOSynt's choreography decentralization, goal recognition represents a fundamentally different problem regarding realizability en-

choreos.disim.univaq.it) extends and advances their preliminary version.[6,7]

Step 5. The generated CDs, together with the description of the services, serve as input to the enactment engine for deployment. CD deployment descriptors are codified in XML.

Step 6. As we mentioned before, CDs are interposed among the participant services needing coordination. CDs coordinate the services' interaction such that the resulting collaboration realizes the specified choreography. To achieve correct coordination, CDs exchange additional communication (codified in XML) at run time to prevent undesired interactions. 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, CDs support the correct execution of the choreography by realizing the required distributed coordination logic among the discovered services.

The Synthesis Processor

As Figure 3 shows, the CHOReOSynt architecture comprises four REST (Representational State Transfer) services that perform the synthesis-time activities in Figure 2. The figure also shows CHOReOSynt's main functionalities.

The M2M transformer. The M2M transformer offers a REST operation `bpmn2clts()`, which implements an ATL-based transformation. It takes as input the BPMN2 choreography specification and automatically generates a *choreography labeled transition system* (CLTS). A CLTS is a finite-state automaton that, for coordination purposes, is suitably extended with fork and

forcement. That is, goal recognition concerns learning a goal-based model of an agent by observing the agent's actions while interacting with the environment. In contrast, realizability enforcement produces a decentralized coordination logic out of a task-based specification of the choreography.

Given a set of candidate services offering the desired functionalities, the TCP-Compose* algorithm identifies the set of composite services that best fit the user-specified qualitative preferences over nonfunctional attributes.[6] CHOReOSynt could exploit this research to extend the discovery process to enable more flexible selection of services from the registry.

The research we described in the main article is an advance over our previous research.[7,8] Although the synthesis process described in our previous research treated most BPMN2 (Business Process Model and Notation Version 2.0) constructs, it considered a simplified version of their actual semantics. For instance, as in Güdemann and his colleagues' research, the selection of conditional branches was simply abstracted as a nondeterministic choice, regardless of the run-time evaluation of their enabling conditions. Analogously, the synthesis process enforced parallel flows by nondeterministically choosing one of their linearizations obtained through interleaving, thus losing the actual degree of parallelism.

To overcome these limitations, CHOReOSynt relies on a choreography model that, being more expressive than the choreography model in CIGAR and TCP-Compose*, preserves the BPMN2 constructs' actual semantics. Relying on a more expressive model lets us define a novel, more effective distributed coordination algorithm.

References

1. M. Güdemann, G. Salaün, and M. Ouederni, "Counterexample Guided Synthesis of Monitors for Realizability Enforcement," *Automated Technology for Verification and Analysis*, LNCS 7561, Springer, 2012, pp. 238–253.
2. R. Kazhamiakin and M. Pistore, "Choreography Conformance Analysis: Asynchronous Communications and Information Alignment," *Web Services and Formal Methods*, LNCS 4184, Springer, 2006, pp. 227–241.
3. R. Kazhamiakin and M. Pistore, "Analysis of Realizability Conditions for Web Service Choreographies," *Formal Techniques for Networked and Distributed Systems—FORTE 2006*, LNCS 4229, Springer, 2006, pp. 61–76.
4. M. Trainotti et al., "ASTRO: Supporting Composition and Execution of Web Services," Service-Oriented Computing—ICSOC 2005, LNCS 3826, Springer, 2005, pp. 495–501.
5. D. Hao Hu and Q. Yang, "CIGAR: Concurrent and Interleaving Goal and Activity Recognition," *Proc. 23rd Nat'l Conf. Artificial Intelligence* (AAAI 08), 2008, pp. 1363–1368.
6. G.R. Santhanam, S. Basu, and V. Honavar, "TCP-Compose*—a TCP-Net Based Algorithm for Efficient Composition of Web Services Using Qualitative Preferences," *Service-Oriented Computing—ICSOC 2008*, LNCS 5364, Springer, 2008, pp. 453–467.
7. M. Autili et al., "A Model-Based Synthesis Process for Choreography Realizability Enforcement," *Fundamental Approaches to Software Engineering*, LNCS 7793, Springer, 2013, pp. 37–52.
8. M. Autili, A. Di Salle, and M. Tivoli, "Synthesis of Resilient Choreographies," *Software Engineering for Resilient Systems*, LNCS 8166, Springer, 2013, pp. 94–108.

join constructs, conditional branching, and conditional loops (typical BPMN2 constructs). The CLTS provides CHOReOSynt with a formal choreography model independent from the specific choreography modeling notation. This means you can use CHOReOSynt in different practical contexts in which different modeling notations might be adopted, provided you've implemented a dedicated M2M transformation.

Starting from the CLTS, the M2M transformer extracts the participant list, using `extractParticipants()`, and derives the CLTSs modeling each participant's expected behavior. To produce the participant-specific CLTS, the transformer projects the choreography onto the participant's role, thus filtering out the transitions and related states not belonging to the role description. The participant-specific CLTS models the interaction behavior that a candidate service (to be discovered out of the registry) must support to play the participant's role.

The discovery manager. The synthesis process and the discovery process interact to retrieve, from the registry, candidate services suitable for playing the required participant roles—those services whose behaviors are compatible with the participant-specific CLTS. For each participant, the discovery manager interacts with the extensible service discovery (XSD) system by invoking `discoverServices()`, which takes the participant-specific CLTS as input.

The behavior simulator. Using `simulation()`, the behavior simulator selects the candidate services that can behaviorally simulate a participant-specific CLTS. It takes as input the participant-specific CLTS and the BPEL description of the service retrieved by the XSD service. It

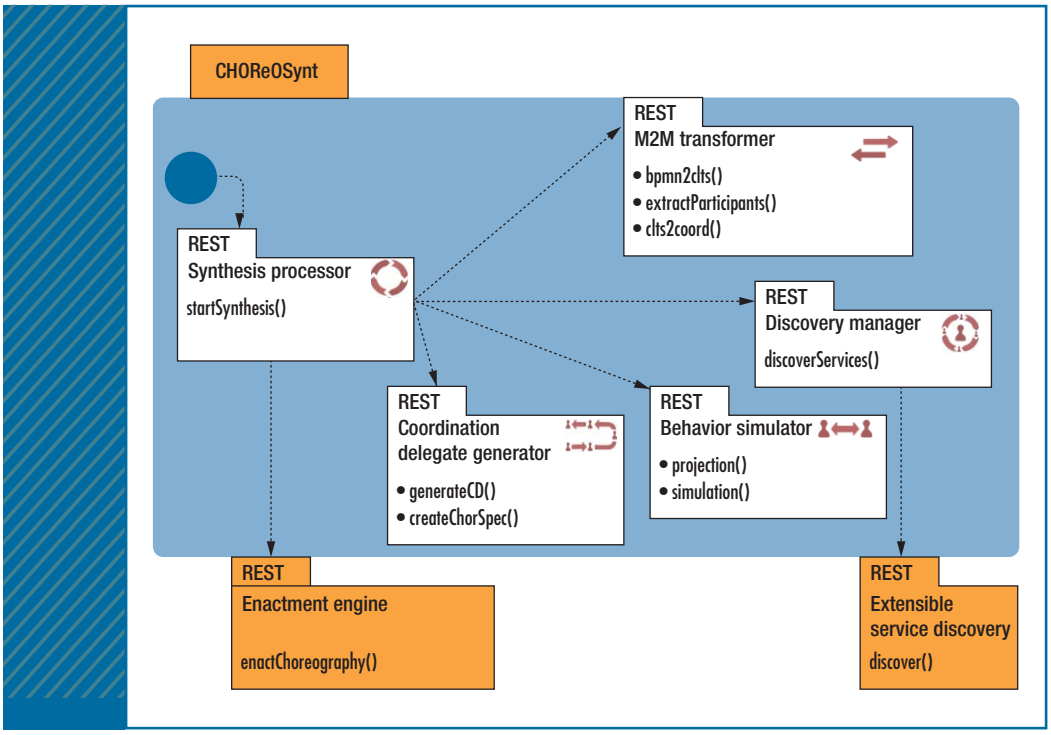

FIGURE 3. The synthesis processor architecture. The four REST (Representational State Transfer) services perform the synthesis-time activities in Figure 2.

implements a form of LTS simulation 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. This means that the discovered service's behavior must cover the participant's expected behavior.

The CD generator. After the services have been selected, the CD generator generates the CDs, using **generateCD()**. This produces an executable description of the choreography to pass to the enactment engine through **createChorSpec()**. The enactment engine takes as input the selected services and the CDs generated for them. In particular, the choreography executable description is an XML-based declarative description specifying the selected services' locations, the generated CDs' locations, and the service–CD and CD–CD interdependencies.

The CHOReOSynt Eclipse Plug-in

CHOReOSynt has two modalities. The user might choose whether to execute the *automatic synthesis processor* or *interactive synthesis processor*. The former produces all the artifacts in one step. The latter produces them step by step and visualizes them by using a graphical editor we developed that's based on GMF (Graphical Modeling Project; www.eclipse.org/modeling/gmp). The synthesis plug-in can be automatically installed into the Eclipse platform through the update site, http://choreos.disim.univaq.it/updatesite/site.xml. A video demonstrating CHOReOSynt at work on the reference scenario is at choreos.disim.univaq.it/downloads.

Our experiments demonstrated that CHOReOSynt can be effectively applied in practical contexts. In particular, they show that considering domain-specific interaction patterns mitigates the complexity of coordination enforceability when recurrent business protocols must be enforced. Generally, choreography synthesis is difficult in that not all possible collaborations can be automatically realized. This suggests we could improve CHOReOSynt with a combination of domain-specific choreography patterns, as well as protocol interaction patterns that correspond to service collaborations that are tractable through exogenous coordination. This approach would also let us produce parameterized coordination patterns offline, which could then be instantiated at run time.

Currently, CHOReOSynt supports pure coordination. It doesn't deal with heterogenous interaction protocol adaptation because it doesn't account for mismatches at the level of service operations and related I/O parameter types. To support data-based coordination through the elicitation and application of complex data mappings, CHOReOSynt should be enhanced to automatically infer mappings to match the data types of messages sent or received by mismatching participant services. This

VIDEO

means effectively coping with heterogeneous service interfaces and dealing with as many enterprise integration patterns[8] and protocol mediation patterns[9] as possible, in a fully automatic way. Toward that end, we achieved promising results in automated synthesis of modular mediators.[10]

We want to enable the market acceptance and further enhancement of CHOReOSynt by third-party developers, especially small and medium enterprises, including development of applications for commercialization. So, we released CHOReOSynt under the umbrella of the Future Internet Software and Services Initiative (FISSi; www.ow2.org/view/Future_Internet/). Using a market-oriented approach, FISSi aims to develop awareness of OW2 Future Internet software in both FISSi members and nonmembers and both open source vendors and proprietary vendors. Our primary objective, to be achieved in the near future, is to establish a community of developers and third-party market stakeholders (for example, users, application vendors, and policy makers) around CHOReOSynt.

Acknowledgments

This research is supported by the Italian Ministry of Education, Universities and Research, project 2012E47TM2 (Integrated Design and Evolution of Adaptive Systems), and by the European Community (Seventh Framework Program, 2007–2013) under grant agreement 257178 (the CHOReOS project).

References

1. D. Calvanese et al., "Automatic Service Composition and Synthesis: The Roman Model," *IEEE Data Eng. Bull.*, vol. 31, no. 3, 2008, pp. 18–22.
2. M. Trainotti et al., "ASTRO: Supporting Composition and Execution of Web Services," *Service-Oriented Computing—ICSOC 2005*, LNCS 3826, Springer, 2005, pp. 495–501.
3. S. Basu, T. Bultan, and M. Ouederni, "Deciding Choreography Realizability," *Proc. 39th Ann. ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages* (POPL 12), 2012, pp. 191–202.
4. S.T.Q. Jongmans, F. Santini, and F. Arbab, "Partially Distributed Coordination with Reo," *Proc. 22nd Euromicro Int'l Conf. Parallel, Distributed and Network-Based Processing* (PDP 14), 2014, pp. 697–706.
5. I. Lanese, F. Montesi, and G. Zavattaro, "Amending Choreographies," *Proc. 9th Int'l Workshop Automated Specification and Verification of Web Systems* (WWV 13), 2013, pp. 34–48.
6. M. Autili et al., "A Model-Based Synthesis Process for Choreography Realizability Enforcement," *Fundamental Approaches to Software Engineering*, LNCS 7793, Springer, 2013, pp. 37–52.
7. M. Autili, A. Di Salle, and M. Tivoli, "Synthesis of Resilient Choreographies," *Software Engineering for Resilient Systems*, LNCS 8166, Springer, 2013, pp. 94–108.
8. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley, 2004.
9. X. Li et al., "A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation," *Proc. 7th Working IEEE/IFIP Conf. Software Architecture* (WICSA 08), 2008, pp. 137–146.
10. P. Inverardi and M. Tivoli, "Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns," *Proc. 2013 Int'l Conf. Software Eng.* (ICSE 13), 2013, pp. 3–12.

ABOUT THE AUTHORS

MARCO AUTILI is an assistant professor in the University of L'Aquila's Department of Information Engineering, Computer Science, and Mathematics. 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. Autili received a PhD in computer science from the University of L'Aquila. Contact him at marco.autili@univaq.it.

PAOLA INVERARDI is a full professor in the University of L'Aquila's Department of Information Engineering, Computer Science, and Mathematics. Her research focuses on software specification and verification of concurrent and distributed systems, deduction systems, and software architectures. Inverardi received an honorary PhD in computer science from Mälardalen University. Contact her at paola.inverardi@univaq.it.

MASSIMO TIVOLI is an assistant professor in the University of L'Aquila's Department of Information Engineering, Computer Science, and Mathematics. His research focuses on component adaptation and coordination, connector synthesis, software model elicitation, and choreography synthesis. Tivoli received a PhD in computer science from the University of L'Aquila. Contact him at massimo.tivoli@univaq.it.

See www.computer.org/software-multimedia for multimedia content related to this article.