Distributed Composition of Highly-Collaborative Services and Sensors in Tactical Domains

Alexander Perucci(✉), Marco Autili, Massimo Tivoli, Alessandro Aloisio, and Paola Inverardi

Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy
{alexander.perucci,marco.autili,massimo.tivoli, alessandro.aloisio,paola.inverardi}@univaq.it

Abstract. Software systems are often built by composing services distributed over the network. Choreographies are a form of decentralized composition that models the external interaction of the services by specifying peer-to-peer message exchanges from a global perspective. When third-party services are involved, usually black-box services to be reused, actually realizing choreographies calls for exogenous coordination of their interaction. Nowadays, very few approaches address the problem of actually realizing choreographies in an automatic way. These approaches are rather static and are poorly suited to the need of tactical domains, which are highly-dynamic networking environments that bring together services and sensors over military radio networks. In this paper, we describe a method to employ service choreographies in tactical environments, and apply it to a case study in the military domain.

1 Introduction

In the last decade, the Service Oriented Architecture (SOA) paradigm has been highly developed and applied in very different Information Technologies (IT) areas. SOA promotes the construction of systems by composing software services deployed on different devices, connected to the network.

Today's service composition mechanisms are based mostly on service orchestration, a centralized approach to the composition of multiple services into a larger application. Orchestration works well in rather static environments with predefined services and minimal environment changes. These assumptions are inadequate in tactical domains, which are characterized by highly-dynamic networking environments that brings together services and sensors over low-availability military radio networks with low-performing execution nodes. In contrast, service choreography is a decentralized composition approach that was recognized in the BPMN2 (Business Process Model and Notation Version 2.0[1] – the de facto standard for specifying choreographies), which introduced dedicated choreography-modeling constructs through *Choreography Diagrams*. These diagrams model

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

P. Ciancarini et al. (Eds.): SEDA 2018, AISC 925, pp. 232–244, 2020.
https://doi.org/10.1007/978-3-030-14687-0_21

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 only one entity is in charge of centrally determining how to reach the overall objective through centralized coordination of all the involved services.

Recent studies [1,12,17,19–21] have challenged the problem to apply SOA in a tactical domain, providing soldiers with services and sensors in order to better face common difficulties of a mission. Two main aspects are concerned: (i) disadvantaged networks, and (ii) the distributed nature of the tactical field. For example, networks available for land-based military operations suffer from problems like lack of connectivity guarantee, changing network topology that implies no guaranteed service delivery, and radio silence that implies no guaranteed data delivery. A fully distributed service composition approach appears then fitting the tactical environment. Hence, choreographies are a good candidate in that distributed deployment and peer-to-peer communication are appropriate for low-availability networks with low-performing execution nodes.

In this paper, we describe a method to employ service choreographies in tactical environments. Specifically, we report on the extension to the Tactical Service Infrastructure (TSI) that we proposed in the TACTICS project[2] with the approach to the automatic synthesis of choreographies that we proposed in the CHOReVOLUTION project.[3] A case study[4] in the military domain is used to describes our method at work.

The paper is structured as follows. Section 2 sets the context and Sect. 3 introduces a case study in the military domain. Section 4 describes our method at work on the case study. Section 5 discusses related work, and conclusions and future work are given in Sect. 6.

2 Setting the Context

This section sets the context of our work by introducing the projects TACTICS (Sect. 2.1) and CHOReVOLUTION (Sect. 2.2). Then, Sect. 2.3 describes the extension to the TACTICS TSI node that we propose in order to achieve tactical choreographies.

2.1 TACTICS

The TACTICS project studied a way to apply the SOA paradigm over tactical networks, in order to allow information exchange and integration with Command and Control (C2) systems [3,11], and Command, Control, Communication, Computers and Intelligence (C4I) systems [10] for land based military operations.

[2] https://www.eda.europa.eu/info-hub/press-centre/latest-news/2017/06/20/tactics-project-completed.

[3] www.chorevolution.eu.

[4] https://github.com/sesygroup/tactical-choreographies.

It addressed this problem through the definition and the experimental demonstration of the TSI enabling tactical radio networks (without any modifications of the radio part of those networks) to participate in SOA infrastructures and provide, as well as consume, services to and from the strategic domain independently of the user's location. The TSI provides efficient information transport to and from the tactical domain, applies appropriate security mechanisms, and develops robust disruption- as well as delay-tolerant schemes. This includes the identification of essential services for providing both a basic (core) service infrastructure and enabling useful operational (functional) services at the application level. As just mentioned, an interesting goal of the project was to provide an experimental setup (including a complete network setup using real radio communication, and running exemplary applications and operational services) to demonstrate the feasibility of the concepts in a real-life military scenario.

2.2 CHOReVOLUTION

The CHOReVOLUTION project developed the technologies required to implement dynamic choreographies via the dynamic and distributed coordination of (possibly, existing third-party) services. The overall goal was to implement a complete choreography engineering process covering all the development activities, from specification to code synthesis, to automatic deployment and enactment on the Cloud. The outcome was the CHOReVOLUTION Platform[5] together with an integrated development environment realized as a customization of the Eclipse platform called CHOReVOLUTION Studio.[6]

2.3 Tactical Choreographies

As already anticipated, the CHOReVOLUTION results were leveraged to extend the TACTICS TSI node. The TSI node is the core of the TSI. It is essentially a middleware that supports the consumption of remotely provided functional services, interoperating with existing Information Systems and Bearer Systems (i.e., radios). The main goal of the TSI node is to allow connection among services over disadvantaged tactical networks. A service can send a message to another service being agnostic of the problems (i.e., constraints) typical for this kind of networks.

The main functionalities of the TSI node are depicted in Fig. 1. The eight functionalities on the left-hand side belong to three middleware layers: *Service Mediator*, *Message Handler*, and *Packed Handler*. The seven functionalities on the right-hand side are cross-layer, so they can be used in every part of the TSI node. Briefly, the *Service Mediator* manages the user's session, adapts messages to the network bandwidth, and deals with the awareness of the context. The

[5] www.chorevolution.eu/bin/view/Documentation/.
[6] www.chorevolution.eu/bin/view/Documentation/Studio.

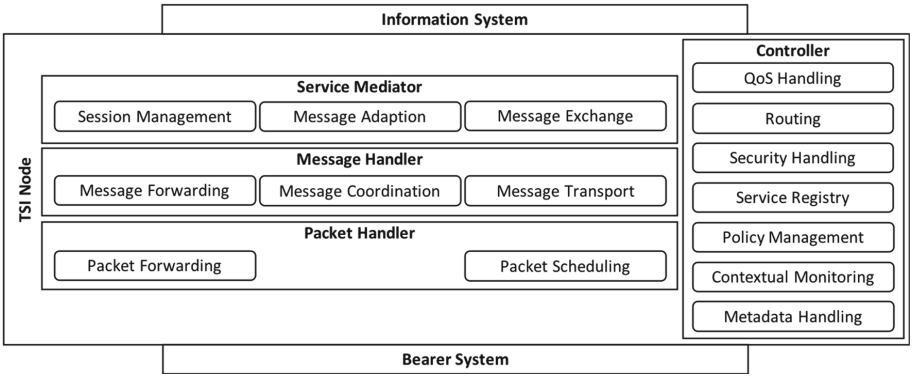

Fig. 1. TSI node

Message Handler stores and/or forwards a message to the next node of a route. Messages are stored in order to cope with network faults. If a message does not reach the destination, the *Message Handler* can resend it. Finally, it deletes a stored message once it receives a receipt confirmation. The *Packet Handler* is mainly used to forward IP packets between different radio networks. Providing detailed description of all the TSI node layers is outside of the scope of this paper. Interested readers can refer to [1,12] for details. The focus of this paper is on the *Message Handler* layer that was extended by adding the new functionality *Message Coordinator* in order to support choreographies. This new functionality is offered by a set of software entities called *Coordination Delegates* (CDs). These entities permit to handle the coordination of messages among the services running on TSI nodes in a way that the desired choreography is fulfilled. When interposed among the services according to a predefined architectural style (a sample instance of which is shown in Fig. 2), CDs proxify the participant services to coordinate their interaction, when needed.

As detailed in Sect. 4, CDs are automatically synthesized, out of a BPMN2 choreography specification, by exploiting the CHOReVOLUTION methodology. CDs guarantee the collaboration specified by the choreography specification through distributed protocol coordination [7]. CDs perform pure coordination of the services' interaction (i.e., *standard communication* in the figure) in a way that the resulting collaboration realizes the specified choreography. To this purpose, the coordination logic is extracted from the BPMN2 choreography diagram and is distributed among a set of *Coordination Models* (CMs) that codify coordination information. Then, at run time, the CDs manage their CMs and exchange this coordination information (i.e., *additional communication*) to prevent possible *undesired interactions*, i.e., those interactions that prevent the choreography realizability [4–7]. 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 on TSI nodes, CDs support the correct execution of the choreography by realizing the required distributed coordination

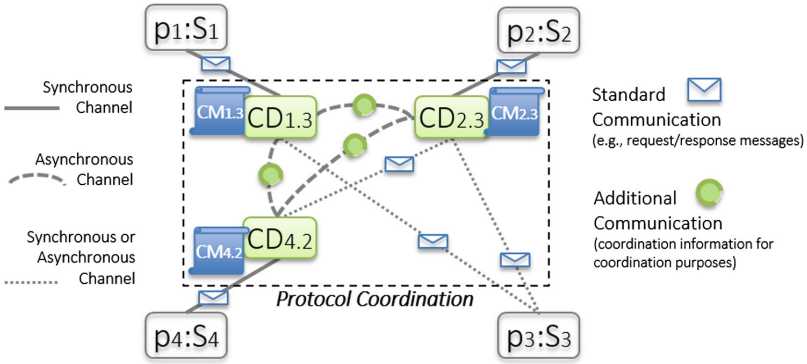

Fig. 2. Architectural style (a sample instance of)

logic among the participant services. Formal foundations of the whole approach and details of the distributed coordination algorithm can be found in [6,7].

3 Case Study

This section introduces a case study in the military domain inspired by the scenario in [2,18]. It concerns an instance of an Emergency Deployment System (EDS) [22], which supports the distributed management and deployment of personnel in cases of natural disasters, search-and-rescue efforts, and military crises. The system gathers information from the tactical environment and gets knowledge of the current status, e.g., the locations and status of the personnel, vehicles, and obstacles.

Considering the scenario represented in Fig. 3, our instance of EDS involves three main subsystems, i.e., Headquarter (HQ), Team Leader (TL), and Responder (RE):

- **HQ** is in charge of both planning and executing operations, in constant communication with TLs in its area and with other HQs in different areas of the same conflict. By gathering information from the field, HQ is fully aware of the current field status and displays on a map the locations and the status of TLs, troops, vehicles and obstacles of interest (both sensed from the environment or pointed out by personnel on the field). Enemy troops and vehicles locations are also displayed as communicated by REs to TLs. That is, HQ is able to send orders to and receive reports from the TLs and to analyze a log of the battlefield status.
- **TLs** are responsible for a smaller part of the field in which they lead the deployment of the REs. They gather information from the REs so to maintain and analyze the current status of the field they are responsible for, and forward it to the HQ in command and to the other TLs in the same area.
- **REs** receive direct orders from TLs. Their main function is to sense the environment and report the status of the location they are deployed in.

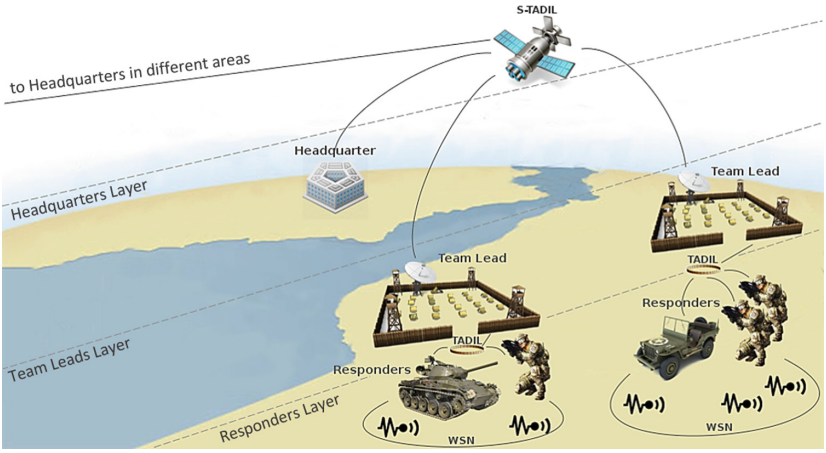

Fig. 3. EDS scenario

Conventional military operations are usually composed of three main activities:

- Reconnaissance – to sense the environment and report to the HQ;
- Command and Control – to plan and organize a strategy by delivering orders to assigned forces;
- Effect – to execute orders in the field and deliver reports to the HD.

Figure 4 shows the portion of the overall system architecture that concerns the services involved in a reconnaissance activity, according to the tasks flow specified by the choreography in Fig. 5.

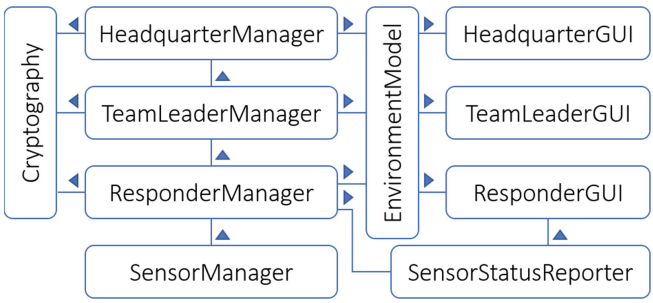

Fig. 4. System architecture related to the choreography in Fig. 5

In BPMN2, a choreography *task* is an atomic activity that represents an interaction by means of one or two (request and optionally response) message exchanges between two participants. Graphically, BPMN2 choreography diagrams use rounded-corner boxes to denote choreography tasks. Each of them

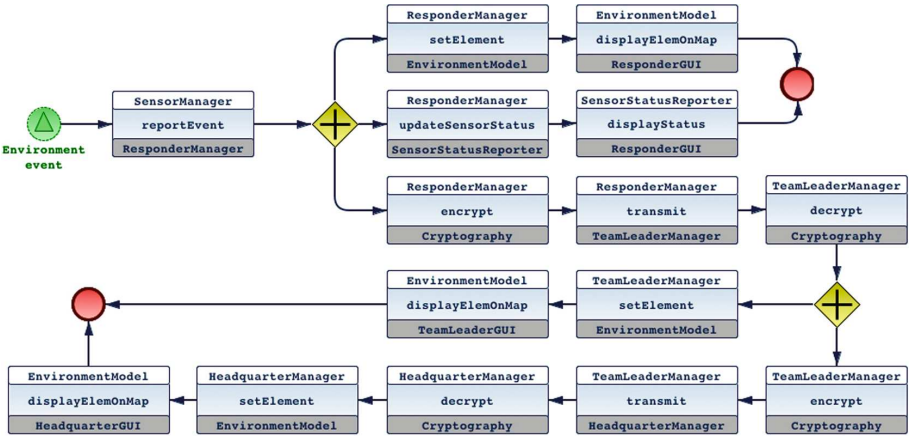

Fig. 5. BPMN2 choreography specification

is labeled with the roles of the two participants involved in the task (e.g., *SensorManager* and *ResponderManager*), and the name of the service operation (e.g., *reportEvent*) performed by the initiating participant and provided by the other one. A role contained in the white box denotes the initiating participant (e.g., *SensorManager* of the first task on the left). In particular, the BPMN2 standard specification employs the theoretical concept of a token that, traversing the sequence flows and passing through the elements in a process, aids to define its behavior. The start event generates the token that must eventually be consumed at an end event. For the sake of presentation, in Fig. 5, input/output messages are not shown although they are internally handled by the synthesis processor according to their XML schema type definition. The tactical choreography is triggered by events associated to variation of elements in the environment. Upon receiving an event, the `SensorManager` reports it to the `ResponderManager` by invoking the *reportEvent* operation. The `ResponderManager` reacts by creating three parallel flows in order to:

1. display on the `ResponderGUI` dedicated graphical markers to locate on a map the elements detected in the environment. For this purpose, the `ResponderManager` calls the *setElement* operation provided by the `EnvironmentModel` service, which in turn calls the *dispalyElemOnMap* operation provided by `ResponderGUI` to show the updated map on the RE's device.
2. display the current status of the sensors on the `ResponderGUI`. In order to accomplish this task, the `ResponderManager` calls the *updateSensorStatus* operation provided by the `SensorStatusReporter` service, which in turn calls *displayStatus* to update the sensors status on the RE's device.
3. transmit the detected elements to its TL – see the *transmit* operation. Before transmitting the message containing the information of the detected elements, the `ResponderManager` encrypts the message by using the `Cryptography`

service. Upon receiving the message, the TL decrypts it, and creates two parallel flows to:

3.1 update the map on the TL device: after the detected elements have been set through the *setElement* operation, the `EnvironmentModel` calls the *dispalyElemOnMap* provided by the `TeamLeaderGUI` service;

3.2 transmit the encrypted element updates to its HQ: upon receiving the updates, in order to display them on the `HeadquarterGUI`, the `HeadquarterManager` follows the same steps as the ones performed by the TLS and REs.

4 Method at Work

The CHOReVOLUTION methodology uses a synthesis processor that takes as input the BPMN2 choreography diagram in Fig. 5, and derives the intermediate automata-based model in Fig. 6, called Choreography-explicit Flow Model (CeFM). This model makes explicit coordination-related information that in BPMN2 is implicit, hence representing an intermediate model suitable for the automated synthesis of the CDs and their CMs. For instance, the state S14 models the fork state in Fig. 5 creating the two parallel flows described by the items 3.1 and 3.2 in Sect. 3.

As already done for the choreography diagram, we do not show the input/output messages of the operations, and make use of abbreviations for role names by showing only capital letters.

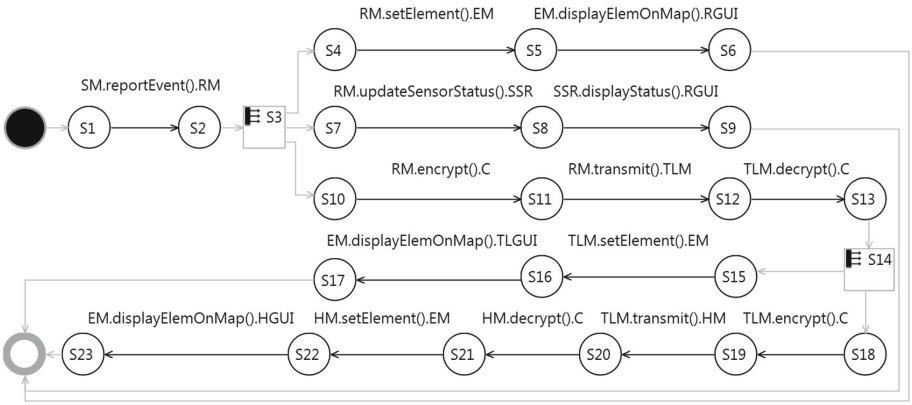

Fig. 6. CeFM derived from the choreography in Fig. 5

Table 1 shows the CMs synthesized from the CeFM of Fig. 6. We refer to [7] for a formal definition of the CeFM construction process and CMs synthesis. For the purposes of our case study, it is sufficient to say that coordination information are codified into the CMs as a set of tuples. For each tuple $\langle s, t, s', CD_{s'}, \rho, Notify_s, Wait_{siblings(s)} \rangle$:

Table 1. CMs tuples derived from the CeFM in Fig. 6

$\mathbf{CM_{EM,HGUI}}$	$\mathbf{CM_{EM,RGUI}}$
$\langle S22, displayElemOnMap(), S23, \{\}, true, \{\}, \{\}\rangle$	$\langle S5, displayElemOnMap(), S6, \{\}, true, \{\}, \{\}\rangle$
$\langle S23, \varepsilon, FINAL, \{\}, true, \{\}, \{\}\rangle$	$\langle S6, \varepsilon, FINAL, \{\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{EM,TLGUI}}$	$\mathbf{CM_{SSR,RGUI}}$
$\langle S16, displayElemOnMap(), S17, \{\}, true, \{\}, \{\}\rangle$	$\langle S8, displayStatus(), S9, \{\}, true, \{\}, \{\}\rangle$
$\langle S17, \varepsilon, FINAL, \{\}, true, \{\}, \{\}\rangle$	$\langle S9, \varepsilon, FINAL, \{\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{HM,C}}$	$\mathbf{CM_{HM,EM}}$
$\langle S20, decrypt(), S21, \{(HM, EM)\}, true, \{\}, \{\}\rangle$	$\langle S21, setElement(), S22, \{(EM, HGUI)\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{RM,C}}$	$\mathbf{CM_{RM,EM}}$
$\langle S10, encrypt(), S11, \{(RM, TLM)\}, true, \{\}, \{\}\rangle$	$\langle S4, setElement(), S5, \{(EM, RGUI)\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{RM,SSR}}$	$\mathbf{CM_{RM,TLM}}$
$\langle S7, updateSensorStatus(), S8, \{(SSR, RGUI)\}, true, \{\}, \{\}\rangle$	$\langle S11, transmit(), S12, \{(TLM, C)\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{SM,RM}}$	$\mathbf{CM_{TLM,C}}$
$\langle S1, reportEvent(), S2, \{\}, true, \{\}, \{\}\rangle$	$\langle S12, decrypt(), S13, \{\}, true, \{\}, \{\}\rangle$
$\langle S2, \varepsilon, S3, \{\}, true, \{\}, \{\}\rangle$	$\langle S13, \varepsilon, S14, \{\}, true, \{\}, \{\}\rangle$
$\langle S3, \varepsilon, S10, \{(RM, C)\}, true, \{\}, \{\}\rangle$	$\langle S14, \varepsilon, S15, \{(TLM, EM)\}, true, \{\}, \{\}\rangle$
$\langle S3, \varepsilon, S4, \{(RM, EM)\}, true, \{\}, \{\}\rangle$	$\langle S14, \varepsilon, S18, \{(TLM, C)\}, true, \{\}, \{\}\rangle$
$\langle S3, \varepsilon, S7, \{(RM, SSR)\}, true, \{\}, \{\}\rangle$	$\langle S18, encrypt(), S19, \{(TLM, HM)\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{TLM,EM}}$	$\mathbf{CM_{TLM,HM}}$
$\langle S15, setElement(), S16, \{(EM, TLGUI)\}, true, \{\}, \{\}\rangle$	$\langle S19, transmit(), S20, \{(HM, C)\}, true, \{\}, \{\}\rangle$

- s denotes the CeFM source state from which the related CD can either perform the operation t or take a move without performing any operation (i.e., the CD can step over unlabelled arrows). In both cases, s' denotes the reached target state;
- $CD_{s'}$ contains the set of (identifiers of) those CDs whose supervised services became active in s', i.e., the ones that will be allowed to require/provide some operation from s'. This information is used by the "currently active" CDs to inform the set of "to be activated" CDs (in the target state) about the changing global state;
- ρ is a boolean condition whose validity has to be checked to select the correct tuple, and hence the correct flow(s) in the CeFM; if no condition is specified, ρ is always set to $true$.
- $Notify_s$ contains the predecessor of a join state that a CD, when reaching it, must notify to the other CDs in the parallel flow(s) of the same originating fork. Complementary, $Wait_{siblings(s)}$ contains the predecessors of join states that must be waited for.

For instance, the first tuple in $CM_{EM,HGUI}$ specifies that $CD_{EM,HGUI}$ can perform the operation *displayElemOnMap* from the source state $S22$ to the target state $S23$; whereas, the second tuple specifies that $CD_{EM,HGUI}$ can step over $S23$ and reach the final state. The first tuple in $CM_{HM,C}$ specifies that $CD_{HM,C}$ can perform the operation *decrypt* from the source state $S20$ to the target state $S21$, and afterwards activates $CD_{HM,EM}$ on the new global state $S21$. At this point, as specified by the corresponding tuple, $CD_{HM,EM}$ can perform the operation *setElement* from $S21$.

The CDs are deployed on the TSI nodes, and at runtime they use the synthesized CMs to control the service interactions enforcing the realizability of the specified choreography.

5 Related Work

The work described in this paper is related to approaches and tools for automated choreography realization.

In [14], the authors propose an approach to enforce synchronizability and realizability of a choreography. The method implementing the approach is able to automatically generate monitors, which act as local controllers interacting with their peers and the rest of the system in order to make the peers respect the choreography specification. Our notion of CD is "similar" to the notion of monitor used in [14], since CDs are able to interact with the choreography participants in order to fulfill the prescribed global collaboration. However, the two synthesis methods are different. In [14], the monitors are generated through an iterative process, automatically refining their behavior.

In [13], the authors address the realizability problem based on a priori verification techniques, using refinement and proof-based formal methods. They consider asynchronous systems where peers communicate via possibly unbounded FIFO buffers. The obtained asynchronous system is correct by construction, i.e., it realizes the choreography specification. With respect to our method and other methods discussed in this section, this method is more scalable in terms of number of involved peers and exchanged messages. However, our approach focuses on realizing a choreography specification by reusing third-party peers (possibly black-box), rather than generating the correct peers from scratch. This is why we cannot avoid to deal with exogenous coordination by means of additional software entities such as the CDs.

The approach in [16] checks the conformance between the choreography specification and the composition of participant implementations. This approach permits to characterize relevant properties to check a certain degree of realizability. The described framework can model and analyze compositions in which the interactions can be asynchronous and the messages can be stored in unbounded queues and reordered if needed. Following this line of research, the authors provided a hierarchy of realizability notions that forms the basis for a more flexible analysis regarding classic realizability checks [15]. The approach statically checks realizability but does not automatically enforce it at run time as we do in our approach.

In [8], the authors identify a class of systems where choreography conformance can be efficiently checked even in the presence of asynchronous communication. This is done by checking choreography synchronizability. Differently from us, the approach in [8] does not aim at synthesizing the coordination logic, which is needed whenever the collaboration among the participants leads to global interactions that violate the choreography specification.

The ASTRO toolset supports automated composition of Web services and the monitoring of their execution [23]. ASTRO deals with centralized orchestration-based processes rather than fully decentralized choreography-based ones.

In [9], the authors present a unified programming framework for developing choreographies that are correct by construction in the sense that, e.g., they ensure deadlock freedom and communication safety. Developers can design both protocols and implementation from a global perspective and, then, correct endpoint implementations are automatically generated. Similarly to us, in [9] the authors consider the notion of multiparty choreography and defines choreography projections. Differently from us, the approach in [9] is not reuse-oriented in that the goal is the generation of the service endpoints' code, rather than the generation of the code for coordinating existing endpoints.

6 Conclusions and Future Work

In this paper, we exploited the results of both the TACTICS project and the CHOReVOLUTION project in order to employ choreographies in a tactical domain. Tactical environments are characterized by low-availability networks with low-performing execution nodes. Preliminary results show that choreographies are a good candidate in that distributed deployment and peer-to-peer communication specifically suite these environments. As future work, we plan to extend the approach to deal with choreography evolution, so to also support choreographies that can vary depending on context changes, intermitting services availability, and user preferences.

Acknowledgment. This research work has been supported by the EU's H2020 Programme, GA No. 644178 (project CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), by the Ministry of Economy and Finance, Cipe resolution No. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies), and by the EDefence Agency, contract No. B 0980 IAP4 GP (project TACTICS - TACTICal Service Oriented Architecture).

References

1. Aloisio A, Autili M, D'Angelo A, Viidanoja A, Leguay J, Ginzler T, Lampe T, Spagnolo L, Wolthusen SD, Flizikowski A, Sliwa J (2015) TACTICS: tactical service oriented architecture. In: 3rd international conference in software engineering for defence applications (SEDA), pp 1–9
2. Andersson J, de Lemos R, Malek S, Weyns D (2009) Modeling dimensions of self-adaptive software systems. In: Software engineering for self-adaptive systems, volume 5525 of LNCS. Springer, Heidelberg, pp 27–47
3. Australian Defence Force Warfare Centre (2009) ADDP 00.1 Command and Control - Department of Defence, CANBERRA ACT 2600

4. Autili M, Di Ruscio D, Di Salle A, Inverardi P, Tivoli M (2013) A model-based synthesis process for choreography realizability enforcement. In: Fundamental approaches to software engineering, volume 7793 of LNCS. Springer, Heidelberg, pp 37–52
5. Autili M, Inverardi P, Tivoli M (2015) Automated synthesis of service choreographies. IEEE Softw 32(1):50–57
6. Autili M, Inverardi P, Tivoli M (2018) Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates. Sci Comput Program 160:3–29
7. Autili M, Tivoli M (2014) Distributed enforcement of service choreographies. In: Proceedings of the 13th international workshop on foundations of coordination languages and self-adaptive systems (FOCLASA), pp 18–35
8. Basu S, Bultan T (2011) Choreography conformance via synchronizability. In: Proceedings of the 20th international conference on World Wide Web (WWW), pp 795–804
9. Carbone M, Montesi F (2013) Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Proceedings of the 40th annual ACM SIGPLAN-sigact symposium on principles of programming languages, pp 263–274
10. National Research Council (1999) Realizing the Potential of C4I: Fundamental Challenges. The National Academies Press
11. Department of Defense (2010) Department of Defense Dictionary of Military and Associated Terms - Joint Publication 1-02 - as amended through 15 December 2014
12. Diefenbach A, Ginzler T, McLaughlin S, Sliwa J, Lampe TA, Prasse C (2016) TACTICS TSI architecture: a European reference architecture for tactical SOA. In: International conference on military communications and information systems (ICMCIS), pp 1–8
13. Farah Z, Ait-Ameur Y, Ouederni M, Tari K (2017) A correct-by-construction model for asynchronously communicating systems. Int J Softw Tools Technol Transf 19(4):465–485
14. Güdemann M, Salaün G, Ouederni M (2012) Counterexample guided synthesis of monitors for realizability enforcement. In: Automated technology for verification and analysis, volume 7561 of LNCS. Springer, Heidelberg, pp 238–253
15. Kazhamiakin R, Pistore M (2006) Analysis of realizability conditions for web service choreographies. In: Formal techniques for networked and distributed systems (FORTE). Springer, Heidelberg, pp 61–76
16. Kazhamiakin R, Pistore M (2006) Choreography conformance analysis: asynchronous communications and information alignment. In: WebServices and formal methods, volume 4184 of LNCS. Springer, Heidelberg, pp 227–241
17. Lopes RRF, Wolthusen SD (2015) Distributed security policies for service-oriented architectures over tactical networks. In: IEEE military communications conference (MILCOM), pp 1548–1553
18. Malek S, Beckman N, Mikic-Rakic M, Medvidovic N (2005) A framework for ensuring and improving dependability in highly distributed systems. In: Architecting dependable systems III, volume 3549 of LNCS. Springer, Heidelberg, pp 173–193
19. Małowidzki M, Dalecki T, Bereziński P, Mazur M, Skarżyński P (2016) Adapting standard tactical applications for a military disruption-tolerant network. In: International conference on military communications and information systems (ICMCIS), pp 1–5

20. Seifert H, Franke M, Diefenbach A, Sevenich P (2012) SOA in the CoNSIS coalition environment: extending the WS-I basic profile for using SOA in a tactical environment. In: Military communications and information systems conference (MCC), pp 1–6
21. Suri N, Morelli A, Kovach J, Sadler L, Winkler R (2015) Agile computing middleware support for service-oriented computing over tactical networks. In: IEEE 81st vehicular technology conference (VTC Spring), pp 1–5
22. Tahmoush D, Lofland C (2009) A prototype national emergency deployment system. In: IEEE conference on technologies for homeland security (HST), pp 331–338
23. Trainotti M, Pistore M, Calabrese G, Zacco G, Lucchese G, Barbon F, Bertoli P, Traverso P (2005) ASTRO: Supporting composition and execution of web services. In: Service-oriented computing - ICSOC 2005, volume 3826 of LNCS. Springer, Heidelberg, pp 495–501