Received: 14 March 2021 | Revised: 10 November 2021 | Accepted: 23 November 2021

DOI: 10.1002/smr.2416

SPECIAL ISSUE - METHODOLOGY PAPER

Journal of Software: Evolution and Process WILEY

Synthesis of context-aware business-to-business processes for location-based services through choreographies

Gianluca Filippone | Marco Autili | Massimo Tivoli

Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy

Correspondence
Gianluca Filippone, Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Vetoio, 67100 L'Aquila, Italy.
Email: gianluca.filippone@graduate.univaq.it

Abstract

Modern technologies and emerging wireless communication solutions in the Information and Communications Technology (ICT) world are empowering the spread of the most disparate ready-to-use software services distributed over the globe and accessed by an increasing number of users. This state of affairs encourages the development of systems based on the reuse of existing services through composition approaches, notably choreographies. Also Public Administrations are driven towards a digitalization process which exploits composition approaches to build complex and interoperable systems that can be remotely accessed by citizens and authorities. However, an automatic support is needed in order to realize the service composition and the distributed coordination logic that enforces the correct choreography realization. Moreover, the need for building dynamic and user-centered systems calls for the realization of choreographies capable to adjust their behavior to the surrounding context. This work presents our proposal for addressing the choreography realization problem, by describing an automated process for the synthesis of choreography-based systems. The synthesized systems are location-aware and able to adapt the services' interaction according to the user's needs and context conditions. We show and evaluate our approach at work on a real use case scenario in the Public Administration domain.

KEYWORDS
adaptation, architecture synthesis, business-to-business processes, context-awareness, coordination, service choreographies

1 | INTRODUCTION

The current trend in the internet and communication fields of Information and Communications Technology (ICT) is represented by the spread of new technologies and solutions which are leading to a fully connected world populated by an increasing number of mobile devices, whose incorporated sensors and communications facilities, together with the availability of a virtually infinite number of software services, are contributing to the digitalization of our "always connected" daily life.[1-4] An increasing number of people uses internet everyday to access services that permit to gain benefits from unprecedented digital functionalities, which are dematerializing business and bureaucratic processes. Still, modern communication technologies like 5G are enabling the coexistence on the network of a very wide range of software services, which can be involved in many different everyday life scenarios. Smart cities represent one of the most important outcomes of this process. The interaction between services and connected things, like intelligent infrastructures, smart vehicles, and sensors, allows smart cities to deliver a large variety of new services to citizens and public authorities.[5-8]

1.1 | Motivations, challenges, and opportunities

The above setting encourages the development of added-value applications that can be conveniently built by reusing and composing existing (ready to use) services geographically distributed throughout the territory and scattered in different locations. In fact, by following the smart cities vision, also local administrations are taking steps to make their services accessible to citizens and other public authorities. The challenge here is to be able to compose different services, offered by different authorities, knowing that it is not always possible (or convenient) to centralize responsibilities; rather, a multiparticipant peer-style perspective is the only way forward.

Given the above, choreographies represent a powerful and flexible service composition approach in which participant services are loosely coupled and their interaction is performed without any central entity.[9-12] Services communicate in a peer-to-peer style (without the asymmetry found in, e.g., client-server style or orchestration based approaches), autonomously take decisions and, out of the blue, engage in the interaction by performing tasks according to their imminent needs and local state. Thus, choreographies are particularly useful in those situations where multiple processes of different organizations/authorities need to collaborate but none of them want to, or cannot, takes full responsibility for performing a centralized coordination. This is exactly what happens in Business-to-Business (B2B) applications, which are cross-enterprise by definition and the involved parties have often comparable negotiating or decisional power.

An important observation here is that, technically, a centralized approach based on, for example, service orchestration, can in principle be adopted to realize a system whose interactions, at the end, accomplish the same goal as the choreography-based one. Clearly, this can be done at the price of a loss in "performance" due to an inevitably reduced parallelism, further accounting for the introduction of a single point of failure. In any case, the above considerations basically concerns a problem of governance that goes beyond and prevents the adoption of a centralized approach, regardless.

1.2 | Working context

The ConnectPA Italian project collects[1] the needs and the innovation requests of the Italian Public Administration (PA). The project goal is to develop a choreography-based approach for the provisioning of IT solutions for public administrations by implementing a platform capable of interconnecting, composing and coordinating geographically distributed services in order to create dynamic and interoperable e-Government services for smart cities. In the ConnectPA context, modern PA systems are realized as a composition of both existing services and new ones, which cooperate in order to allow the interoperation between different public authorities. Thanks to this approach, different PAs can dynamically interoperate, hence easing and speeding up complex administrative B2B processes.

1.3 | Problem space

When considering scenarios in which third-party services are reused and composed as black-box entities, the realization of the distributed coordination logic, which is needed to achieve the correct interaction, may be a complex and error-prone activity.[9,10] The services involved in the choreography act as active entities which concurrently perform tasks and autonomously take decisions. They may not synchronize as prescribed by the choreography, and the global collaboration may not follow the specification. Composing such services and realizing the required *distributed coordination* calls for a suitable automated support that aids the development activities by providing correct-by-construction and ready-to-use solutions to concurrency and realizability issues. Moreover, in the ConnectPA context, local institutions like municipalities can offer more complex value-added services by composing their own local services through choreographies and make them accessible by citizens depending on their location. Thus, *location-awareness* represents a crucial context dimension, and the resulting choreography-based system needs to be aware of user's location in order to let her access the right services, at the right moment, in the current location.

1.4 | Solution space

In previous works,[9,10] choreography realization is enforced through the automated synthesis of additional software entities, called *Coordination Delegates* (CDs) that, when interposed between the participant services, control their interactions by running a distributed coordination algorithm and exchanging synchronization messages. In this paper, we propose an enhanced version of CDs, called *context-aware Coordination Delegates* (caCD) that are able to dynamically "sense" the context conditions and adapt the choreography behavior accordingly. We describe our approach

[1] https://www.maggioli.com/connectpa

at work on a real use case we have developed within the ConnectPA project. In discussing the use case, we show how our approach is able to perform choreography adaptation by dynamically selecting the requested services according to the current user's location.

1.5 | Novel contribution

This work thoroughly extends the approach proposed in our preliminary work for the automated synthesis of context-aware service choreographies.[13] There,[13] context-awareness is only dealt with at design time through the a priori specification of all possible choreography variations. As novel advances, in this paper we:

- propose a novel solution to the realization of context-aware choreographies that makes use of a context model in order to define the context, its acquisition and evaluation;
- present the novel notion of context-aware CDs that now dynamically realize context-aware coordination and adaptation, capable of correctly accessing at run time and coordinating the involved services according to the current context conditions;
- show our solution at work on a use-case designed for the ConnectPA project, which shows the composition and the interoperation of different services in the PA;
- report on experimental data.

1.6 | Paper structure

The paper is structured as follows. Section 2 briefly recalls background notions coming from our previous approach that are needed to fully understand our novel proposal and discusses the challenges related to context-aware choreographies. Section 3 presents our reference scenario. Section 4 presents our solution for the automated synthesis of context-aware choreographies. Section 5 describes our approach for the enforcement of choreography execution. Section 6 shows our approach at work on the reference scenario and shows the experimentation results. Related works are discussed in Section 7. Section 8 concludes by also discussing future research directions.

2 | SETTING THE "CONTEXT"

In this section, we recall the main notions coming from our approach for the automated composition of service choreographies, and we discuss context awareness issues, together with the main challenges that arise when dealing with context-aware systems ad choreographies.

2.1 | Choreography diagrams, automated choreography synthesis, and coordination delegates

The OMG's BPMN 2 standard[2] offers Choreography Diagrams, a practical notation for specifying choreographies that, following the pioneering BPMN process and collaboration diagrams, is amenable to be automatically treated and transformed into actual code. BPMN2 choreography diagrams focus on specifying the message exchanges among the participants from a global point of view. A participant role models the expected behavior (i.e., the expected interaction protocol) that a concrete service should be able to perform to play the considered role. Thus, in BPMN2, a choreography is modeled as a set of tasks that represent the interactions among the participants services. A task is an atomic activity that describes the message exchanges (request and optionally response) between two participants. Each task is performed by an initiating participant, which sends a message to the receiving participant and, optionally, receives a response message when the task is complete. With reference to the upper side of Figure 1, graphically, a task is represented by a rounded-corner box, and it is labelled with the roles of participant services and the name of the task. The initiating participant role is contained into a white box. Tasks are connected by an arrow, which represent a sequence flow.

The approach to the automated composition and enforcement of service choreographies that is considered and extended in this paper makes use of a synthesis tool which automatically generates the distributed coordination logic of the choreography. A *synthesis processor* takes as input the BPMN2 choreography diagram and the set of services selected as participants. A set of additional software entities, called *Coordination Delegates* (CDs), are automatically generated and properly interposed among the participant services. They proxify the service interaction in order to realize the specified choreography by running a distributed coordination algorithm.[9,10,14] CDs are generated for each *consumer* and *prosumer*

[2]https://www.omg.org/spec/BPMN/2.0.2/.

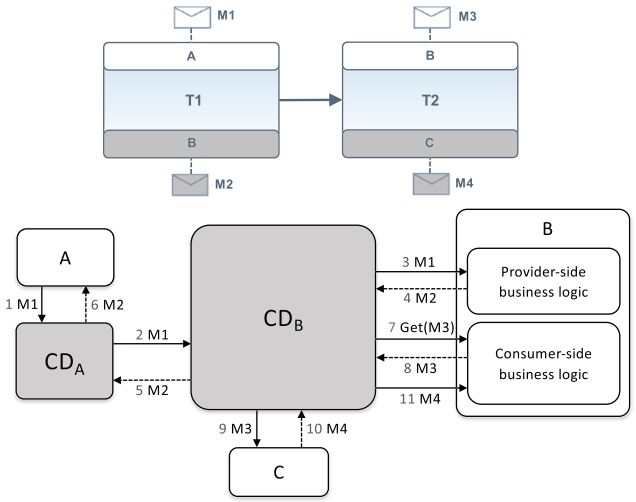

FIGURE 1 Coordination delegates interaction pattern

service[3]. They enforce the correct choreography realization ensuring that the services interaction performs the flows of message exchanges that are prescribed by the choreography specification.[14] CDs allows developers to decouple the coordination logic, that is external to the involved services since it resides in the CDs implementation, from the business logic that instead resides in the services implementation.

Figure 1 shows the CDs interaction pattern by considering the most general sequence of two tasks involving a consumer (A), a provider (C), and a prosumer (B). The prosumer logic is split into *provider-side* and *consumer-side* business logic: the former offers provided functionalities of the service when acting as a provider, and can possibly be implemented by reusing an existing service; the latter represents the logic that is needed to build correct message instances whenever the service requires a functionality from a different service (i.e., when acting as a consumer), hence performing an invocation of it. As it is shown in the figure, the CD generated for the consumer A (CD_A) receives the message sent by the service A (M1) as first, hence forwarding it to the CD generated for the prosumer B, which in turn forwards it to the provider side of B (interactions 1 to 3). Then, the related response messages are sent back, hence ending the execution of task T1 (interactions 4 to 6). In order to execute T2, CD_B asks the consumer-side of B for the message to be sent (M3 in the interactions 7 and 8), forwards it to C, intercepts the response and sends it back (interactions 9 to 11).

The automated synthesis of the choreography is performed through a model-to-code transformation which takes as input the participant models of the choreography. A participant model describes the interaction protocol of a participant, as a partial view of the choreography which considers only the flows in which a participant (i.e., a service) is involved. This model is still a BPMN2 diagram, which is generated through a Model-to-Model (M2M) transformation (*Choreography Projection*) carried out from the choreography specification.[9,14] The synthesis algorithm is able to generate in a fully-automatic way the set of CDs needed for the choreography coordination. Referring to prosumers, whose structure is well defined according to the interaction pattern described above, the synthesis processor generates their skeleton code, since both the provider-side and the consumer-side business logic are application specific and need to be implemented. However, developers have the only task of filling the skeleton code that realizes the business logic of prosumers according to the application needs, without having to focus on possible coordination and concurrency issues.

2.2 | Context-aware choreographies

Consistently with the definition given by Dey,[15] we consider *context-awareness* as the ability, for software systems, of using context information in order to dynamically adapt according to the user needs. Context-aware systems are able to "sense" the *context* and react to changes by adapting their behavior.[16]

Through years, many definitions of *context* have been given in literature, as many are the application domains in which context can be involved. However, context is often defined by enumerating the characteristics describing the application domain that is considered in each work.[17]

[3]A service that is both a consumer and a provider.

Bauer and Novotny[18] reviewed the notions of context across the literature and grouped all the different dimensions that have been taken into account when representing context in three main categories: (i) *social context*, which includes the information about user identity, preferences and habits, emotional state, social and cultural environment, presence and behavior of other people; (ii) *technology context*, including all the characteristics of the physical and virtual platform on which the system runs, such as computing resources, performances, network; (iii) *physical context*, including all the observable elements of the environment in which the system operates, like the functional environment (e.g., indoor/outdoors, car, home), weather, lighting, time, location, movement. A general definition of context, which can in principle be accepted beyond any specific application domain, can be found in Dey[15] where the context is defined as *"any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"*.

A series of challenges that arise when dealing with context-aware systems specifically concerns the representation of the context, the acquisition of the information, and the overhead introduced for the context management.[19] For instance, the *context model* has to consider the heterogeneity of the sources of information that are exploited in order to acquire context data. In fact, some context data can be acquired from sensors placed in the environment, while other data can be provided by the user or mobile devices that interact with the system, or can be obtained from external sources like databases and web services. Moreover, the context model has to allow to derive higher-level context facts from the raw data that are obtained by, e.g., sensors and other external sources, and to allow reasoning in order to decide whether any adaptation is necessary. In addition, the formalism used to define the context models has to be easily understandable by system designers.[20] Thus, in order to allow context-aware systems to adapt according to the runtime context, a mechanism capable of acquiring information from different sources, delivering it to the system and providing context reasoning functions is needed.[20,21]

With particular reference to *service-oriented architectures* (SOA) and choreographies (which represent the most dynamic approach to SOA), realizing context-awareness means being able to adapt the interactions between participants according to the context conditions. The adaptation has to be autonomous, automatic and has to consider the dynamic nature of the context, which can change continuously during the system runtime.[22] Accordingly, the notion of context-aware adaptation we are considering in this paper goes beyond the pure *reconfiguration* of choreographed systems, which can be realized by modifying the choreography specification and evolving the system accordingly (in response to changes of functional or non-functional requirements[23]). Dynamic adaptation means that the portions of the choreography requiring adaptation may not be fully specified at design time. Rather, the choreography specification must be able to express *variability*, hence allowing to specify where adaptation is required and what are the adaptation possibilities. Then, the variable context-aware portions of the choreography (the ones that realize the adaptation) can be left *underspecified* and be *concretized* at runtime when the context conditions are known. That said, we consider three different levels of runtime adaptation for context-aware choreographies:

- **Message-level adaptation**—to adapt the content of one or more messages exchanged between the participants (*context-aware messages*);
- **Participant-level adaptation**—to select, at runtime, the instance(s) of a participant service that have to be involved in a task. Participant instances are selected among a pool of service instances that can play the participant's role (*context-aware participant instantiation*);
- **Task-level or task-flow-level adaptation**—to select, at runtime, a suitable flow of tasks for a choreography portion that has to be executed, among a set of possible interactions (*choreography variants*).

In order to express choreography variability, when defining task flows that require adaptation, we leverage the modeling solution proposed in our preliminary work.[13] In that work, *variation points* are introduced as new BPMN2 elements that allow to define variable parts in the choreography model. The possible alternative interactions are then designed as choreography variants, which in turn are modeled as sub-choreographies and are associated to the variation point(s). When defining tasks that require context-aware participant instantiation, we express variability by using *multi-instance participant*s for specifying that there exist several related participant services that can be involved and dynamically selected. The list of service instances that can be selected at runtime is kept in a service registry. The registry can be dynamically populated with new service instances as they become available.

Concerning context-aware messages, there is no need to leave something unspecified, since the structure of the message must always be defined in the choreography specification since already the beginning, and the content of the message is always a runtime matter. In this case, it is enough to suitably extend the message construction logic (as specified in Section 2.1) in order to take into account the runtime context.

For both task flows and participants, the concretization of choreography variants and multi-instance participants is realized by selecting, among a set of candidate variants or participant services, those which are suitable according to the runtime context of the system.

From here on, we will refer to both variants and participants as *adaptable entities*. Adaptable entities have to be selected at runtime when the choreography execution reaches a variation point or a task with a multi-instance participant.

In next sections, we will (i) present our reference scenario, (ii) discuss our approach for modeling and handling context as a fundamental element of the process of realizing context-aware choreographies, and (iii) present the extension of the framework for the automated synthesis of context-aware choreographies, that is, for the automatic generation of the context-aware adaptation logic according to the three adaptation levels discussed above.

3 | REFERENCE SCENARIO

The scenario described in this section refers to a real use case implemented in the ConnectPA project and shows how Public Administration (PA) systems can be realized through composition of a number of services exposed by different authorities. The system in our scenario aims to dematerialize and digitalize the whole process, from the request to the payment, for billposting of posters in public spaces of towns.

The "classical" procedure for public posting requires a citizen submitting a request to the municipality. Then, if free posting spaces are available, the municipality sends the bill back to the citizen for enabling the payment. In order to ease and dematerialize this process, municipalities expose their own services that manage the billposting, allowing users to search for the available spaces in the municipality's territory, hence sending the posting request for a set of selected spaces, and receiving the related bill. However, in most cases, the postings involve different municipalities. Even if the municipalities expose their own services, citizens have to repeat the process many times for each municipality, and there is no integration with a unified payment service. For this reason, our system allows the interoperability among the services of different municipalities and other services offered by the PA (e.g., a payment service for the PA).

The user interacts with the system through a mobile application that allows her to pinpoint the geographical area in which she wants to affix posters (by selecting a radius starting from her current position) and specify the duration of the postings. Then, the system interacts with the services that handle the billposting of each involved municipality and retrieves the list of available spaces. The application shows the available spaces to the user so that she can filter them and select the ones in which she wants to affix posters. The selected spaces are then added to a virtual cart. After the user has confirmed the request, the system communicates with the services of each municipality in order to get the payment bills. At the end, the user proceeds with the payment through the mobile app. If the payment service is available, the payment is processed, a receipt is sent back to the user and the confirmation of the payment is sent to the involved municipalities. Otherwise, a payment invoice is sent to the user that can pay the bill off-line. With reference to Figure 2, the system is realized as a composition of the following participants:

- *Mobile APP*: it is the client of the system. It interacts with the other participants for the request of posting spaces, the selection of the spaces and the confirmation of the request. Moreover, it allows the user to pay for the bills required to affix posts.
- *Poster Service*: it is a *prosumer* service that includes some of the business logic of the system. As a provider, it receives the requests from the Mobile APP to get the list of available spaces, the confirmation of the selected ones, and the billings for the payments. As a consumer, it communicates with the services of the municipalities and with the payment service.
- *Payment Service*: it is a *prosumer* service that offers the interfaces that are needed to receive the information for the payments and to pay them. As a consumer, it interacts with the municipalities in order to send the confirmation of the paid bills.
- *Cart*: it is a *provider* service that exposes the interfaces needed to add, remove, and get all the elements of a virtual cart.
- *Municipality Poster Service(s)*: these are the *provider* services, offered by the municipalities, that allow to send requests for the public postings.

Figure 3 shows the BPMN2 diagram of the choreography specifying the way the above mobile app and services must interact. By referring to Figure 1, *Mobile APP* and *Poster Service* represent the participants of the task *Send post request*, and *Mobile APP* is the initiating participant. Rhombuses marked with "+" and with two or more outgoing edges represent parallel gateways, which fork the flow in two or more concurrent flows. A parallel gateway with two or more incoming edges and only one outgoing edge represents the join of parallel flows. Rhombuses marked with "x" and with two or more outgoing edges represent exclusive gateways, which create alternative paths within the choreography. Exclusive gateways with two or more ingoing edges are used to merge alternative paths.

A *multi-instance* marker (i.e., a set of three vertical lines) is added in the participant band of a choreography task when a participant service can participate in the choreography with multiple instances. We recall that we exploit multi-instance participants in order to express *variability* needed for the *participant-level adaptation* when the instances of a participant that have to be involved in a task are not known *a-priori*. This is the case of the *Municipality Poster Service*.

Boxes with thicker border are *variation points*. They represent the underspecified portions of the choreography needed for the *task-level adaptation* when a portion of the choreography to be executed has to be selected at runtime. This is the case of *Payment*, which, according to our

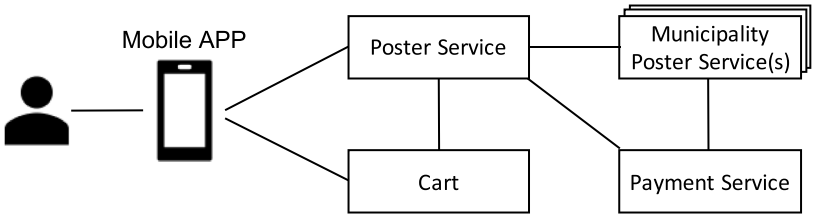

FIGURE 2 System connections

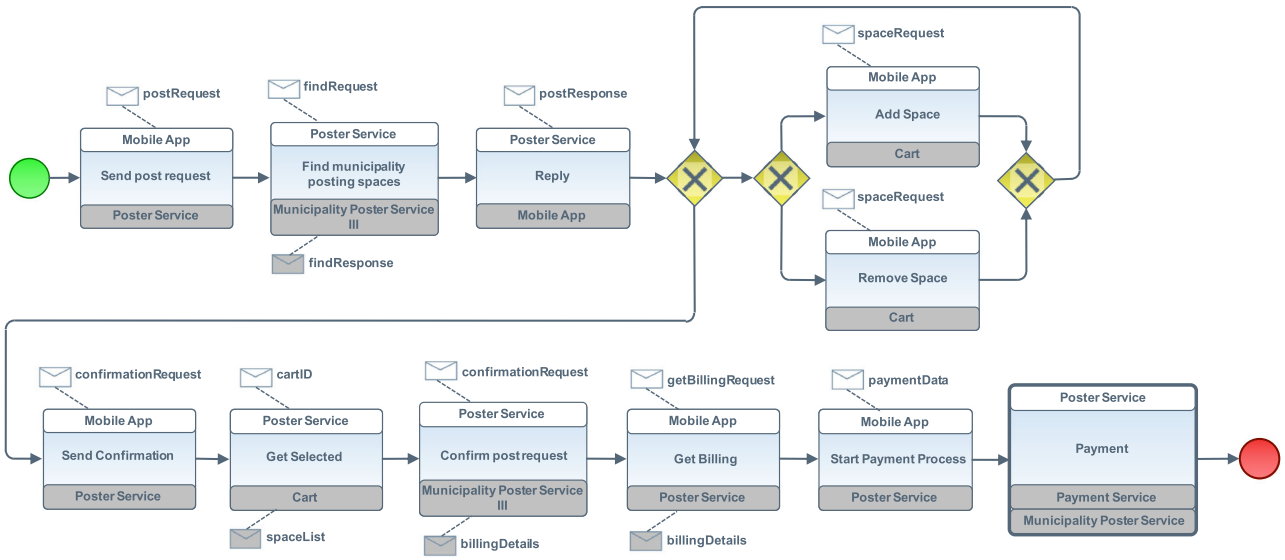

FIGURE 3 Reference scenario: Public billposting choreography

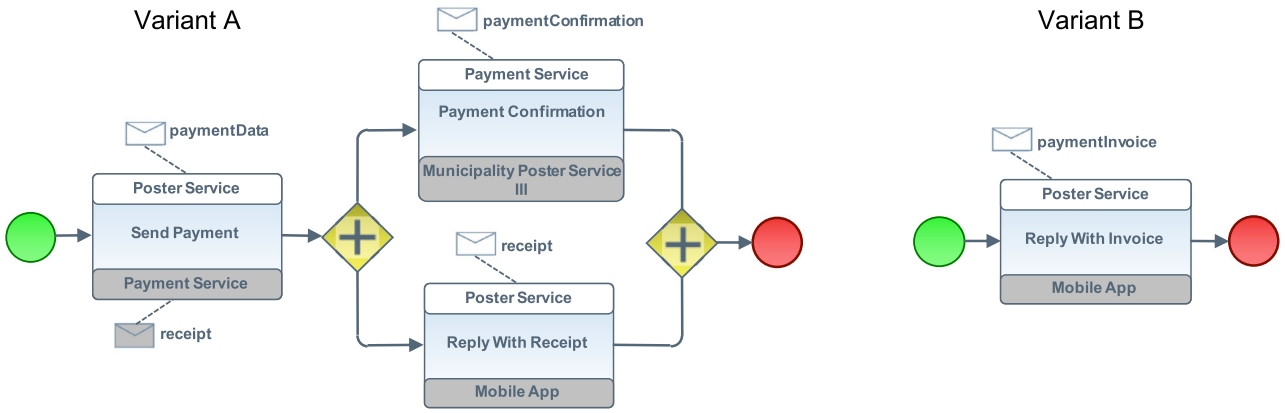

FIGURE 4 Choreography variants for payment variation point

scenario, has to consider the availability of Payment Service. The white label contains the name of the initiating participants of the variants, while the gray labels contains the name other participants. Figure 4 shows the two sub-choreographies that have been defined as *variants*. The first (Variant A) models the interactions that occur if Payment Service is available; the second (Variant B) models the interactions that occur otherwise.

The proposed scenario shows some complexities due to the possibility of having race conditions, the presence of multi-instance participants, choreography variants and independent sequences of message exchanges. Depending on the sequence or timing of the message exchange, a race condition arises when more than one *Mobile App* is simultaneously attempting at gaining free posting spaces from the same municipality. It is easy to see how this kind of race condition is more difficult to manage in the presence of multi instance participants. In fact, multi instance participants require dealing with *multicast* group communication where the related message exchange is addressed to a group of participants "simultaneously". This is the case of the multiple *Municipality Poster Service* instances (see for instance the task *Confirm post order*). Possible race conditions, unknown service instances, independent sequences of message exchanges make things worst. The sequence of tasks *Confirm post request* → *Get Billing* represents an *independent sequence* for which coordination is needed in order to enforce the correct choreography realization and prevent *undesired interactions*. A detailed description of independent sequences and undesired interactions can be found in previous work.[10] Informally, for the purpose of this paper, it is enough to clarify that task sequences are independent if the initiating participant of a task does not appear as participant of the immediately preceding task. In other words, at run time, during the execution of the interested sequences, there is no suitable message exchange between the involved service instances that can "naturally" imply the correct synchronization of their interaction from the inside, according to the choreography specification. As already said, the situation here is more complicated due to the presence of multi instance participants, whose number of instances is not known in advance. Race conditions and independent sequences are then the main cause of

undesired interactions, i.e., those interactions that are not prescribed by the choreography but may occur if the services were left free to interact without any control. That said, considering the reuse of black-box services, and the risk of having undesired interactions, external coordination exploiting additional synchronization messages is required. Moreover, for what concerns the mobile app, another important aspect, which further complicates matters, is that it can autonomously engage in the interaction, out of the blue, in the middle of the independent sequences, without synchronizing with neither the *Poster Service* nor the *Municipality Poster Service*.

Beyond pure coordination issues, the scenario has also context-aware coordination issues. In fact, since each municipality offers its own service for the billposting, the system needs to ask for the availability of posting spaces to the specific (location-based) service that covers the area surrounding the user. For this reason, in our choreography, the *Municipality Poster Services* have been modeled as a multi-instance participant, and a dynamic service selection mechanism is needed in order to correctly access and coordinate the right service at the right location. Also, the request message sent to each service may differ in its content since the set of involved services is not known a priori; rather, it depends on the runtime context (user's location). Moreover, the choreography considers the availability of the Payment Service in order to let the user to complete the billposting process, even in case of unavailability of the payment system. The alternative interaction that can occur according to the availability of the service have been modeled through choreography variants that are associated to a variation point. A mechanism for the dynamic selection and execution of the right variant is needed in order to adapt the choreography behavior to the runtime context (service availability) and coordinate the services interactions in the different scenarios. For these reasons, basic CDs in Autili et al[10] need to be extended in order to handle those cases in which (i) a service or a variant selection needs to be performed at runtime, (ii) the request messages have to be composed according to the runtime context, and then (iii) suitably coordinated with the running choreography. This is where the novel approach described in this paper comes into play and makes the difference by introducing a mechanism for the definition, acquisition and manipulation of the context. A further advance with respect to our previous work is represented by the new notion of *context-aware CDs* that permit to bring together the ability to solve the above described coordination issues together with the context-related ones.

4 | APPROACH DESCRIPTION

The novel approach for the automated synthesis of context-aware choreographies proposed in this paper extends our previous work on choreography synthesis by:

i) Defining a suitable context model;
ii) Realizing means to acquire and manipulate the needed context information;
iii) Implementing a method to select the candidate *adaptable entities* for the context-aware adaptation at runtime, thus concretizing *underspecified* choreography portions;
iv) Executing the resulting adaptation.

Figure 5 overviews the approach. It considers three phases: *system modeling*, *choreography synthesis* and *choreography refinement & enactment*. In the modeling phase, system designers realize the BPMN2 choreography specification, the specification of choreography variants (if any), and the model of the context. These artifacts are taken as input for the synthesis phase, during which a synthesis algorithm performs a model-to-code

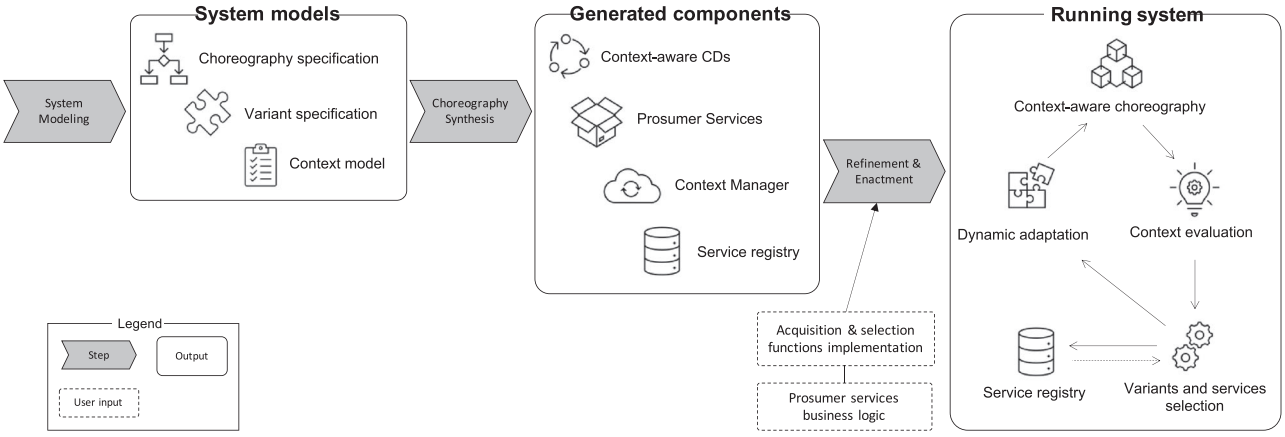

FIGURE 5 Approach overview

transformation and generates: (i) the skeleton code of the prosumer services, (ii) the set of *Context-aware CDs*, (iii) a *Context Manager* service and (iv) a *Service Registry*. Context-aware CDs are an enhancement of our basic CDs. Beyond coordinating the services interactions, they allow to send context-related messages and perform the runtime adaptation by invoking the selected service instances or executing the selected choreography variant. The Context Manager is a service that gets and holds the information about the current context conditions by holding an instance of the context model. The Context Manager acquires the context by receiving *context-carrying messages* or by executing *ad-hoc* written *acquisition functions*, and selects the adaptable entities through *selection functions*. The Service Registry holds the description of all the service instances that can be selected for the tasks with multi-instance participants. The Service Registry offers an interface that is exploited by the Context Manager—in order to get the service descriptions—and by the system administrators—in order to add new service instances to the registry.

The implementation of the generated components is completed in the choreography refinement phase. In this phase, developers provide the business logic of prosumer services and the implementation of acquisition and selection functions of the Context Manager. Then, the choreography can be deployed on a cloud infrastructure and enacted: the resulting system will be able to "sense" the context and to dynamically adapt by dynamically instantiating the underspecified portions of the choreography.

In the following, we will describe how: (i) the context is modeled, (ii) the generated system handles the context information, and (iii) the adaptation is performed. We will also portray some examples from our reference scenario to better explain the process.

4.1 | Context model

The context model allows to define:

- The context of the system and of adaptable entities;
- How the context is acquired;
- How the context is evaluated and how entities are selected.

Representation of the context information—The information that are needed to represent the context can be grouped in two categories:

- *System context*, which includes all the attributes that are common to the whole system (e.g., user preferences, physical, and technological characteristics of the environment, weather, location, availability of services, devices, and common resources);
- *Entities context*, which describes the dynamic conditions of each adaptable entity (e.g., performances and available resources) and their static properties that never change at runtime (e.g., service cost, location reference, and resource requirements).

Figure 6 shows the metamodel that allows to define the context model. The model describes the context as a composition of the metaclasses `SystemContext` and `EntityContext`. The metaclass `ContextAttribute` is used to define the dynamic characteristics of the context of both system and entities, while `StaticProperty` metaclass allows the definition of the static properties of the entities context. Each entity context is associated to an *Entity Class*. The latter represents the set of service instances of a multi-instance participant, or the set of variants associated to a variation point.

Support for context acquisition—In order to deal with the heterogeneity of context sources, our approach considers two different means for acquiring the runtime context data: ad-hoc *acquisition functions*, or *context-carrying messages*.

Acquisition functions allow to acquire context data from any source, like the external environment, devices or web services. They are declared as *Context Source* for a context attribute through the `AcquisitionFunction` metaclass. During the choreography synthesis, the skeleton code of the declared acquisition functions is generated, while their implementation needs to be provided by developers in the choreography refinement phase (Figure 5).

Further information about the context can be found in the content of business messages exchanged by participants. In fact, choreographies may involve devices equipped with sensors, which are able to provide runtime context data. For instance, messages sent by client applications running on smartphones may contain data about user's location. Those messages are already defined into the BPMN2 choreography specification: we call them *context-carrying messages*. Thus, when the value of a context attribute can be acquired from a context-carrying message, the latter can be used as context source instead of defining an acquisition function. The definition of a `ContextCarryingMessageSource` requires to define the *XPath* expression that allows to extract the needed context information from the message. Unlike acquisition functions, the code needed for the acquisition of the context data is generated from the context model and does not need any further refinement.

Support for context evaluation and entities selection—In order to provide support for the evaluation of the context and the selection of the adaptable entities, the metamodel allows to define a *selection function* for each entity class. The definition of a `SelectionFunction` requires the reference to the context attributes that have to be evaluated, and the cardinality of the result. For instance, a single result is required when selecting a variant, while multiple result may be required for the selection of the service instances for a multi-instance participant. As for acquisition functions, during the modeling phase the selection functions are defined, while their implementation needs to be provided in the refinement phase.

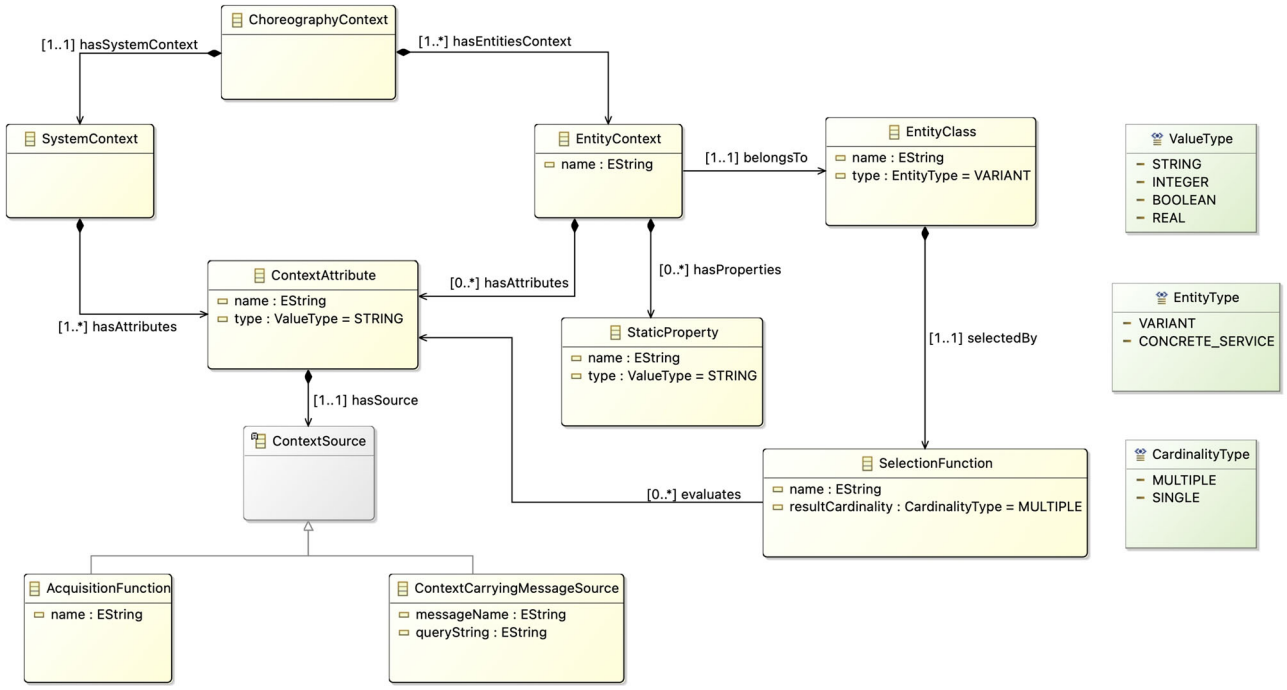

FIGURE 6 Context metamodel

```
Choreography Context
    System Context
        Context Attribute userLatitude [type: REAL]
            ContextCarryingMessageSource [messageName: "postRequest", queryString: "/locationCoordinates/latitude"]
        Context Attribute userLongitude [type: REAL]
            ContextCarryingMessageSource [messageName: "postRequest", queryString: "/locationCoordinates/longitude"]
        Context Attribute searchingRadius [type: REAL]
            ContextCarryingMessageSource [messageName: "postRequest", queryString: "/searchRadius"]
        Context Attribute paymentAvailability [type: BOOLEAN]
            AcquisitionFunction getPaymentAvailability
    Entity Context MunicipalityPSContext [belongsTo: MunicipalityPosterService]
        Static Property referenceLatitude [type: REAL]
        Static Property referenceLongitude [type: REAL]
    Entity Context PaymentVariantContext [belongsTo: PaymentVariant]
        Static Property paymentServiceRequired [type: BOOLEAN]
Entity Class MunicipalityPosterService [type: CONCRETE_SERVICE]
    Selection Function selectMunicipalities [resultCardinality: MULTIPLE, evaluates: userLatitude, userLongitude, searchingRadius]
Entity Class PaymentVariant [type: VARIANT]
    Selection Function selectPaymentVariant [resultCardinality: SINGLE, evaluates: paymentAvailaility]
```

FIGURE 7 Context model for the reference scenario

Figure 7 shows the context model of our reference scenario. The characteristics that constitute the context are the user's position, the selected searching radius, and the availability of the Payment Service. These are modeled into the *System Context* through the context attributes *userLatitude*, *userLongitude*, *searchingRadius* and *paymentAvailability*. Concerning the context sources of these attributes, note that the data about the user's position and the selected searching radius can be provided by the Mobile App, which exploits the user's phone GPS sensors in order to acquire the GPS position, and obtains the searching radius from the user's input. The message *postRequest*, sent by the Mobile App thruogh the task *Send post request*, contains these data. Figure 8 shows the XML schema of the message. *postRequest* can be exploited as context source for the context attributes *userLatitude*, *userLongitude* and *searchingRadius*. Thus, a `ContextCarryingMessageSource` is defined in the context

```
<xsd:complexType name="postRequest">
   <xsd:sequence>
      <xsd:element name="requestID" type="xsd:string"/>
      <xsd:element name="startDate" type="xsd:date"/>
      <xsd:element name="endDate" type="xsd:date"/>
      <xsd:element name="locationCoordinates">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="latitude" type="xsd:double"/>
               <xsd:element name="longitude" type="xsd:double"/>
               </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
     <xsd:element name="locationName" type="xsd:string"/>
     <xsd:element name="searchRadius" type="xsd:integer"/>
   </xsd:sequence>
</xsd:complexType>
```

FIGURE 8 XML schema of the context-carrying message "*postRequest*"

model for each of them, with the related *XPath* string: `/locationCoordinates/latitude` for the user latitude, `/locationCoordinates/longitude` for the user longitude, and `/searchRadius` for the searching radius. In contrast, an acquisition function is defined in order to get the runtime value of *paymentAvailability*: `getPaymentAvailability`. The function allows to check the availability of the service by contacting it.

Since the scenario involves a multi-instance participant and a variation point, two different `EntityClasses` have been defined: *MunicipalityPosterService* and *PaymentVariant*. For each of them, an entity context is defined. *MunicipalityPSContext* defines the properties of the Municipality Poster Service instances. Since they are characterized by the location of the municipality—which never changes at runtime—their entity context is composed by the properties *referenceLatitude* and *referenceLongitude*. *PaymentVariantContext* defines the properties of the variants defined for the variation point. They are characterized by the availability of the Payment Service, i.e., the condition used to select a variant. Thus, their entity context is composed by the *paymentServiceRequired* property. Finally, for each of these two entity classes, a selection function has been defined. For *MunicipalityPosterService*, the selection function *selectMunicipalityServiceInstances* is defined. It evaluates the context attributes related to the user location and the searching radius, and returns a result with a multiple cardinality, since it has to select all the services whose reference location is into the selected area. For the variation point, the function *selectPaymentVariant* is defined. It evaluates the *paymentAvailability* context attribute and returns a single result (the selected choreography variant).

4.2 | Context-aware CDs

Context-aware CDs (caCDs) are introduced to execute the choreography adaptation at runtime. They manage:

- The execution of variants;
- The context-aware participant instantiation;
- The exchange of context-aware and context-carrying messages;
- The communication with the Context Manager.

The whole logic of caCDs is automatically generated.

For "normal" tasks, caCDs behave as basic CDs (see Figure 1), by also exchanging SYNCH messages[10] to coordinate independent sequences of tasks. CaCDs execute an extended version of the interaction pattern of basic CDs. This extension accounts for: caCDs involved in tasks with multi-instance participants; caCDs that handle the execution of a variant; caCDs whose supervised participant sends a context-carrying message. Thus, in the following, we describe the interaction pattern of the caCDs when: (i) receiving context-carrying messages, (ii) performing *context-aware participant instantiation*, and (iii) executing choreography variants. In so doing, for the sake of simplicity, we consider the Context Manager as a black-box entity. Its behavior will be discussed in Section 4.3.

Exchange of context-carrying messages—Figure 9 shows the interaction pattern for a general case represented by a task T1 in which a context-carrying message (message M1) is sent by a *consumer* A to a *prosumer* B. When receiving M1, the $caCD_A$ forwards it (as a basic CD would

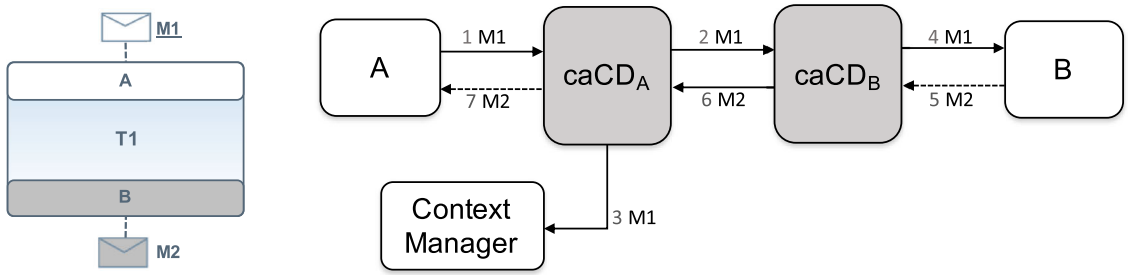

FIGURE 9 Context-aware CD interaction pattern for context-carrying messages

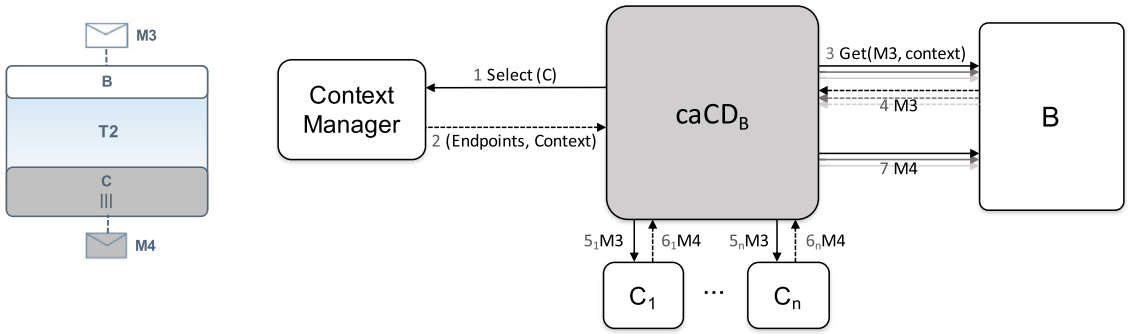

FIGURE 10 Context-aware CD interaction pattern for context-aware participant instantiation

also do, interactions 1 and 2), then it sends M1 to the Context Manager in order to be processed (extension of the basic pattern, interaction 3). When handling the response message, it behaves like a basic CD (interactions 6 and 7).

In our reference scenario, this interaction occurs for the task *Send post request* since it involves the context-carrying message *postRequest*. When receiving this message, the caCD of *Mobile App* forwards it both to the caCD of *Poster Service* and to the Context Manager. The latter will get the values of the context attributes *user latitude*, *user longitude* and *searching radius* from the message content.

Context-aware participant instantiation—Figure 10 shows the interaction pattern for a general task T2 in which a context-aware participant instantiation is needed for a multi-instance participant C. In order to execute the task, the $caCD_B$ asks the Context Manager for selecting the instances of C that have to be involved (interaction 1). The Context Manager returns the set of the endpoints of the selected instances, together with their entities context and the system context (interaction 2). Then, $caCD_B$ will execute the context-aware participant instantiation. For each service instance C_i in the list, $caCD_B$ will: (i) get from B the instance of the message M3, related to the context associated with the instance C_i (interactions 3 and 4); (ii) send the message to C_i, receiving back the response (interactions 5 and 6); (iii) forward the response message to the service B (interaction 7).

The participant instances are invoked in parallel. Thus, interactions from 3 to 7 are concurrently executed for each selected service instance. Note that the message M3 is built by considering the runtime context, which is provided by the Context Manager alongside each selected service instance endpoint. In this way, also the *message-level adaptation* can be realized, as discussed in section 2.2.

In our reference scenario, the context-aware participant instantiation is performed for the tasks *Find municipality posting spaces*, *Confirm post request*, and *Payment confirmation* (in the Variant A of Figure 4). They all require the runtime selection of the instances of *Municipality Poster Service*. For the first two tasks, the interactions described above are performed by the caCD of *PosterService*, since the latter is the initiating participant of the tasks. For the task *Payment confirmation*, the instantiation is handled by the caCD of *Payment Service*.

Variants execution—Figure 11 shows the interaction pattern for the selection and execution of a choreography variant. Here, it is important to know which are the initiating participants of the first task of the variants. That is, A and B in Figure 11. After the execution of T1, the caCD of A ($caCD_A$) asks the Context Manager for the selection of the variant associated to the variation point VP1 (interaction 1). The Context Manager returns the name of the selected variant (interaction 2). Then, if the initiating participant of the selected variant is B, $caCD_A$ communicates with $caCD_B$ in order to start the execution of a variant; otherwise, $caCD_A$ starts the execution of the first task of the selected variant.

In our reference scenario, the selection of a variant (Figure 4) has to be performed for the variation point *Payment*. The caCD of *Mobile App* asks the Context Manager for the variant selection after the execution of *Start Payment Process*. *Variant A* is selected if the payment service is available, *Variant B* otherwise. Since for both of them the initiating participant is *Poster Service*, in both cases the caCD of *Mobile App* informs the

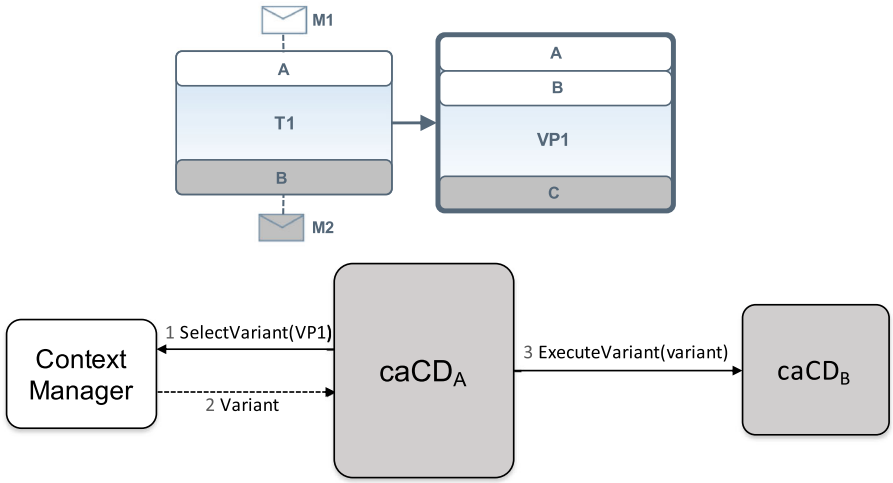

FIGURE 11 Context-aware CD interaction pattern for variant selection

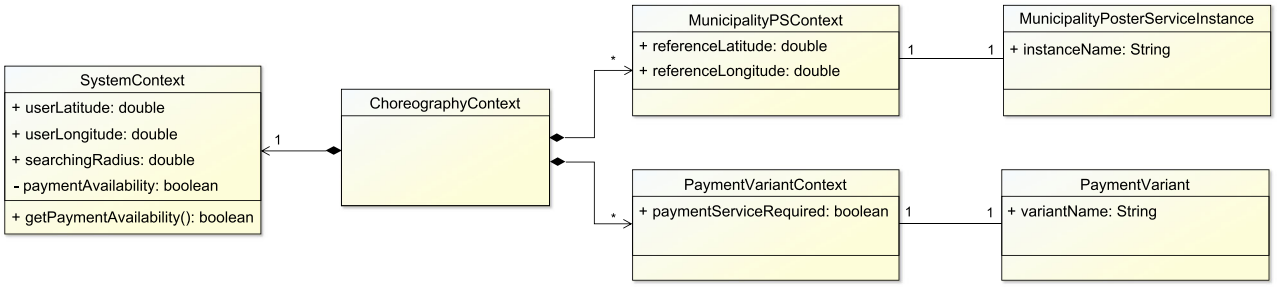

FIGURE 12 Class diagram of the context representation in the Context Manager

caCD of *Poster Service* about which variant has to be executed. Then, if the selected variant is *Variant A*, the caCD starts the task *Send Payment*; *Reply With Invoice*, otherwise.

4.3 | Context manager

The Context Manager holds the model instance of the context of both system and entities, receives the context-carrying messages that are exchanged during the choreography execution, and executes the acquisition and selection functions. In the following, we describe how: (i) the context is represented in the Context Manager; (ii) the context acquisition is performed; (iii) the entities are selected at runtime.

Context model instance—The synthesis process transforms the context model into a set of Java classes, which are instantiated at runtime. The Java classes contain, as class attributes, the context attributes that have been defined in the model. At runtime, there will be:

- An instance of the system context class;
- An instance of an entity context class for each adaptable entity (i.e., an "entity context" instance for each instance of a participant service and for each choreography variant).

The list of service instances are obtained by querying the Service Registry. Moreover, when a new service is published into the registry, the latter will interact with the Context Manager in order to update the list and create a new instance of the entity class context.

Figure 12 shows the class diagram that represents the context of our reference scenario. The context attributes defined for the system context are translated into the attributes of the class `SystemContext`. The acquisition function *getPaymentAvailability* is translated into the method `getPaymentAvailability()`. The class `MunicipalityPSContext` contains the information of the homonymous entity class defined in the context model (Figure 7). The same holds for the `PaymantVariantContext` class. An instance of `SystemContext` will dynamically hold the

runtime information of the system context, while there will be an instance of `MunicipalityPSContext` for each instance of Municipality Poster Service in the registry, and an instance of `PaymentVariantContext` for each of the two variants.

Context acquisition—As explained above, the context information are acquired through context-carrying message or acquisition functions. Once obtained the context information, the Context Manager updates the values of the related attributes into the context model instance.

When a context-carrying message is sent, the involved caCD forwards it to the Context Manager. By means of an aptly defined *XPath* expression, the latter extracts the value of all the context attributes for which the received message has been defined as `ContextCarryingMessageSource`. In contrast, the values of the attributes that have an acquisition function as context source are obtained on demand, when they are used for the entities selection.

Coming back to our scenario, the Context Manager receives the context-carrying message *postRequest* that is used as context source for the attributes *userLatitude*, *userLongitude*, and *searchingRadius*. The Context Manager extracts the value from the message and updates the values of the *userLatitude*, *userLongitude* and *searchingRadius* attributes in the instance of the class *SystemContext*. Instead, the value of the context attribute *paymentAvailability* is obtained by executing its defined acquisition function. This is done before the execution of the selection function of the variants.

Entities selection—When a variation point or a task with multi-instance participants needs to be executed, the Context Manager selects the adaptable entities by executing the selection function. When selecting a variant, the Context Manager simply returns to the involved caCD the name of the selected variant. When dealing with the context-aware participant instantiation, once selected the service instances, the Context Manager obtains their description (i.e., the service endpoints) from the service registry. Then, the Context Manager returns the list of the endpoints of the selected instances, together with their runtime context.

With reference to our scenario, the Context Manager executes the selection function `selectMunicipalityServiceInstances` when the instances of Municipality Poster Service have to be selected. Once obtained the list of selected instances, it gets their endpoints from the service registry and returns them to the caCD associated to the multi-instance participant. In order to perform the adaptation needed for the variation point *Payment*, the Context Manager first executes the acquisition function `getPaymentAvailability`, then executes the selection function `selectPaymentVariant`. The name of the selected variant is then returned to the caCD of Poster Service, which will execute the selected variant.

Note that both the context evaluation and the service selection are performed before that services are involved in the interaction. Thus, instances are selected only when they are "inactive" (i.e., only when it is not in the middle of serving a request), hence their substitution cannot affect the consistency of the overall choreography state. In order to avoid inconsistencies, we assume that:

- the service instances to adapt are stateless, that is, there is no internal state to be restored; rather, there can be a conversational state that is maintained across each single invocation through message passing and/or "external" session handling[4];
- the context conditions that are evaluated for the selection cannot change between an instance selection and the next one (e.g., the context attributes are acquired from context-carrying messages that are sent only once, as in our reference scenario).

Moreover, we also assume that all the service instances of a multi-instance participant share the same interface and the same interaction protocol, thus we do not focus on possible interface incompatibilities. In Section 8, we better discuss how interface incompatibilities can be handled.

5 | ENFORCING CHOREOGRAPHY EXECUTION

As explained in Section 3, besides the issues related to the *context-awareness*, our reference scenario presents some coordination problems due to the presence of *independent sequences* of tasks, that may cause the choreography execution to not follow the specification and, hence, undesired interactions. For these reasons, the interaction pattern of CDs needs further enhancements with respect to the patterns shown in Sections 2.1 and 4.2, by introducing *SYNCH* messages.

Figure 13 shows the most general cause that leads to independent sequences of tasks. For the sake of generality, let us consider Participant A and Participant C be two prosumers. We extend the interaction pattern of basic CDs (Section 2.1) by considering the use of a synchronization message (*SYNCH*) in order to sequentialize the execution of independent tasks, as prescribed by the specification. The SYNCH message is sent from the CD handling the first task ($CD_{Participant\ A}$) to the CD handling the second task ($CD_{Participant\ C}$), as soon as the first task has been accomplished. Note that $CD_{Participant\ C}$ is by construction waiting to receive the SYNCH before performing any other action and, hence, any other possibly concurrent interaction with C will be blocked until it can be performed, according to the specification. The same holds if the CD that receives a SYNCH message is associated to a consumer service. In that case, since the consumer can engage the interaction at any time by sending the

[4]In the case stateful services with internal state, ad-hoc store and restore procedures must be put in place. In Section 8 we discuss in more detail how a suitable notion of "quiescence" can be profitably exploited in this case.

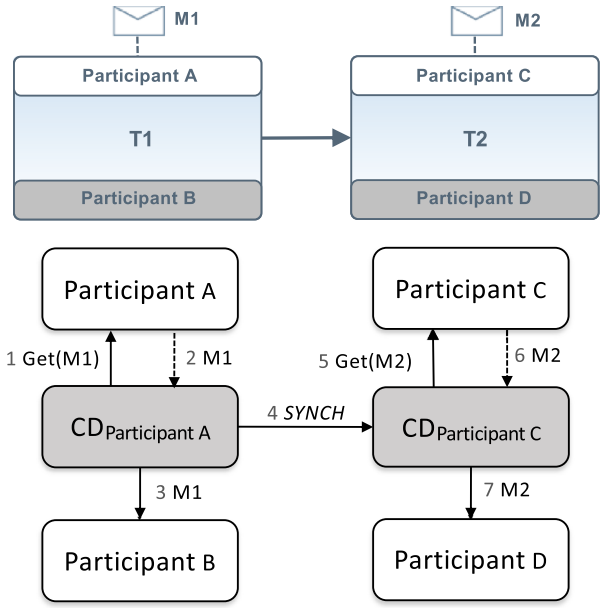

FIGURE 13 Coordination of an independent sequence of tasks

request message to the associated CD, the CD receives it and then waits for the SYNCH message before forwarding the request message to the receiving participant.

Coming back to our reference scenario, the sequence of tasks *Confirm post request* → *Get Billing* is an independent sequence. By following the described approach, the coordination of this sequence is realized by a SYNCH message that is sent from the caCD of *Poster Service* (initiating participant of *Confirm post request*) to the caCD of Mobile App (initiating participant of *Get Billing*). Even if the Mobile App tries to perform the task *Get Billing* before *Confirm post request* is executed, its caCD prevents this (undesired) interaction until the choreography execution reaches the correct state. That is, the state in which *Confirm post request* has been just accomplished.

Our approach is also able to enforce the coordination of more complex cases, when the choreography specification includes independent sequences of tasks that go through a fork or a join gateway and also involve multi-instance participants, as in the case of our reference scenario. In fact, the same coordination mechanism can be used, by sending SYNCH messages from all the CDs that are handling the tasks that precede the gateway(s) to all the CDs that handle the tasks that immediately follow the gateway(s). This solution allows CDs to be aware of the state of the choreography also including tasks in which they are not involved, enforcing the choreography to wait to reach the correct execution state before performing tasks, hence avoiding undesired interactions.

For a formalization—and related proof of correctness—of the synthesis method used to deal with all the possible sources of choreography unrealizability, we refer to our previous work.[10]

6 | EXPERIMENTATION

In this section we describe how the system designed for our use case is realized according to our synthesis framework, and how we set up the environment in order to run experiments. Then we discuss the obtained results. The experimentation purpose is to evaluate that: (i) the choreography execution is correctly enforced by the caCDs, that is, the participant services interact as prescribed by the choreography specification; (ii) caCDs are able to achieve dynamic adaptation of the system with respect to the context; (iii) the distributed coordination and adaptation layer of the system does not have any significant impact on the choreography execution time, even when the number of concurrent requests or the number of adaptable entities grows, i.e., the overhead introduced by the Context Manager and caCDs for the adaptation is negligible.

6.1 | Generated system

The synthesis algorithm takes as input the choreography specification (Figure 3), plus its variants (Figure 4), and the context model (Figure 7).

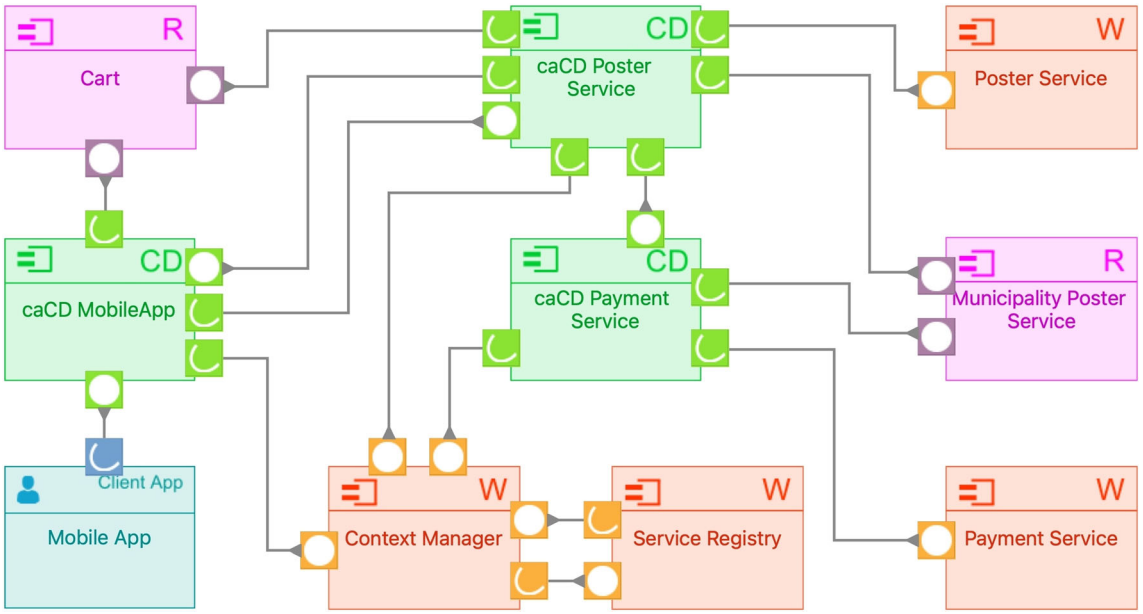

FIGURE 14 Generated system architecture

```
public List<MunicipalityPosterServiceInstance> selectMunicipalities(
      double userLatitude,
      double userLongitude,
      double searchingRadius,
      List<MunicipalityPosterServiceInstance> instances) {

  return instances.stream().filter(e ->
      distance(e.getContext().referenceLatitude, e.getContext().referenceLongitude, userLatitude, userLongitude) <= searchingRadius
  ).collect(Collectors.toList());
}


public PaymentVariant selectPaymentVariant(boolean paymentAvailability, List<PaymentVariant> variants) {

  return variants.stream().filter(e -> e.getContext().paymentServiceRequired == paymentAvailability).findFirst().get();
}
```

FIGURE 15 Selection functions implementation

Figure 14 shows system architecture resulting from the choreography synthesis phase. The synthesis process generates the Context Manager, the Service Registry and the following artifacts:

- *caCD MobileAPP*, associated to the participant Mobile APP.
- *caCD Poster Service*, associated to the participant Poster Service.
- *caCD Payment Service*, associated to the participant Payment Service.
- The skeleton code of the *prosumer* services *Poster Service* and *Payment Service*. The former contains the logics needed to interact with the Mobile App and handle the requests of posting spaces. It needs to be implemented by following the skeleton code that has been generated. The latter can be realized by reusing an existing payment service for its provider-side business logic, and implementing the consumer-side business logic only.

The implementation of the Context Manager has been completed by implementing the selection functions and the acquisition function defined in the model.

Figure 15 shows the implementation of the two selection functions. The methods signatures have been generated by the synthesis algorithm; the bodies of the methods have been manually implemented during the refinement phase. The `selectMunicipalities` method filters the municipalities and returns the ones whose distance from the user's position is less than the searching radius. The distance between two pair of coordinates is computed by using the `distance()` method, which has been aptly coded. The `selectPaymentVariant` method returns the variant that requires the Payment Service, if it is available, the one that does not require the service, otherwise.

The acquisition function `getPaymentAvailability` has been implemented to simulate the interaction with the service and return test values.

6.2 | Experimentation settings

For experimentation purposes, we deployed five different instances of *Municipality Poster Service*, each referring to a different municipality. Each instance holds the information about the posting spaces available in its territory. Figure 16 shows the geographical distribution of the considered five municipalities.

The information about the instances of the Municipality Poster Service have been put into the Service Registry through its API. When the choreography is enacted, the Context Manager initializes the context of each of the instances with the information about the location of the municipalities.

In order to perform tests, the *Mobile App* is realized as an application that simulates the presence of a varying number of users that concurrently interact with the system. For each "simulated" user, the Mobile app sends a request by selecting randomly the current user's position and the searching radius. Then, in order to simulate its autonomous behavior, the Mobile App attempts to perform an interaction 1.5 s after the end of the previous one. That is, for each "simulated" user, the Mobile App invokes the *Add Space* operation 1.5 s after that the*Reply* message is received from Poster Service, then after 1.5 invokes *Send Confirmation*, and so on.

We have executed tests by running the choreography both without concurrency and with an increasing level of concurrency by simulating the number of interacting users. Moreover, we also simulate the presence of a growing number of instances of Municipality Poster Service by replicating the entries in the Service Registry. This has been done to stress both the Context Manager in the selection process and the caCDs in the runtime participant instantiation.

Data are collected by caCDs, Context Manager and participant services, which locally log the start and end timestamps of each task, and are obtained only after that the execution is completed, in a way that the collection of data does not interfere with the message exchange between participants and CDs. Consumer services log the timestamp when sending the request message to the receiving participant (interaction 1 of the interaction pattern in Figure 1), and when receiving the response message, if any (interaction 6 in the figure). Provider services log the timestamps when receiving the request messages and when replying with a response message. Prosumer services log the timestamps when receiving the request messages if they play the role of provider. If they play the role of consumer, they log the timestamp when receiving the request from the CD (interaction 5), and when receiving the response message, if any (interaction 11). The Context Manager logs the timestamps when receiving the request for the selection of and adaptable entity and when replying back the list of selected ones (interactions 1 and 2 in Figures 10 and 11). Context-aware CDs log the timestamp when sending and receiving SYNCH messages.

The experimentation has been performed using six Virtual Machines (VMs) installed in three distinct Server Machines (SMs). Each SM is equipped with 2 CPUs Intel Xeon E5-2650 v3, 2.3 GHz, 64 GB RAM and 1 Gb/s LAN network; each VM has 4 CPU cores and 4 GB of RAM. The

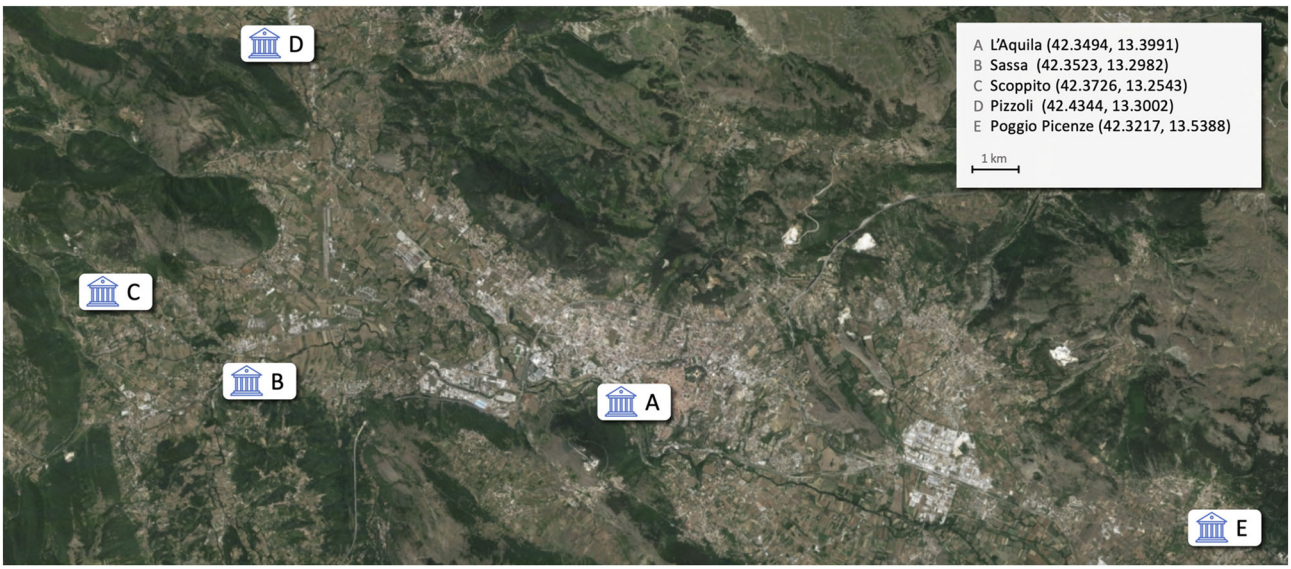

FIGURE 16 Geographical distribution of the municipalities involved in the choreography

operating system is Ubuntu Server 20.10. Open Stack is the cloud infrastructure provider. Participant services, CDs and Context Manager have been deployed on the VMs as described in Table 1.

This deployment setting permitted us to experiment a real scenario, where services are offered by different providers, and also the instances of *Municipality Poster Service* are geographically distributed, hence running simultaneously with real parallelism. The same holds for caCDs, Context Manager and Service Registry.

6.3 | Experiment results

This section reports the results obtained by running the experiments as set in the previous section. Concerning our use case, experimental results show that (i) the choreography execution is enforced in the correct way; (ii) the context-awareness capabilities are successfully realized; (iii) the overhead introduced by caCDs and Context Manager is negligible even with a large number of concurrent requests and selected service instances, and its growth is linear with respect to the increasing of both the number of requests and the number of selected instances.

Choreography execution enforcement—Figure 17 shows a timeline of the average execution times of the tasks when a single user is simulated. For the sake of simplicity, we report the tasks that are executed when Variant A (Figure 4) is selected. For each task, the blue bar represents the time during which the task is executed (i.e., the time between the first and the last logged timestamps). If the task belongs to an independent sequence, a striped bar shows the time between the sending of the message by the initiating participant to its associated CD (interaction 1 in Figure 1) and the forwarding of the message to the receiving participant (interaction 2). For the tasks with multi-instance participants (marked with a "*" in Figure 17) the blue bar shows the average execution times for all the involved instances of Municipality Service.

Tasks are executed by fully respecting the choreography specification. In fact, although Mobile App can try to invoke tasks in a total autonomous way, their execution is enforced to follow the specification. Figure 18 shows the detail of the execution of the independent sequences of the choreography. The rhombus represents the sending time of the SYNCH message sent by the caCD of Poster Service to the caCD of Mobile App, for enforcing the sequence *Confirm post request* → *Get Billing*. We can note that, since Mobile App tries to execute autonomously the task *Get Billing* while *Confirm Post Order* is still being executed, the message is not forwarded until the latter is completed and the SYNCH message is sent (striped portion of the bar). Only after that the SYNCH message is received, the caCD MobileApp forwards the request message and *Get Billing* is effectively executed (filled portion of the bar for *Get Billing*).

Impact on system performances—In order to evaluate the impact of the approach on the performances, we measured:

TABLE 1 Services deployment scheme for the experimentation

Server machine	Virtual machine	Deployed service
SM1	VM1	Context Manager, Municipality A Poster Service
	VM2	Cart, Municipality B Poster Service, Service Registry
SM2	VM3	Poster Service, caCD Poster Service
	VM4	Municipality C Poster Service, caCD Payment Service
SM3	VM5	Municipality D Poster Service, Payment Service
	VM6	Municipality E Poster Service, caCD MobileApp

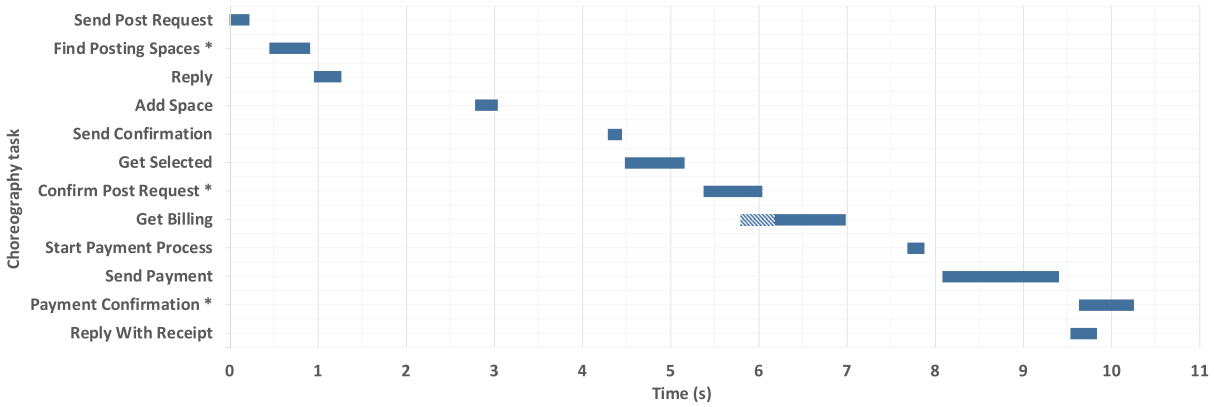

FIGURE 17 Average execution time of choreography tasks

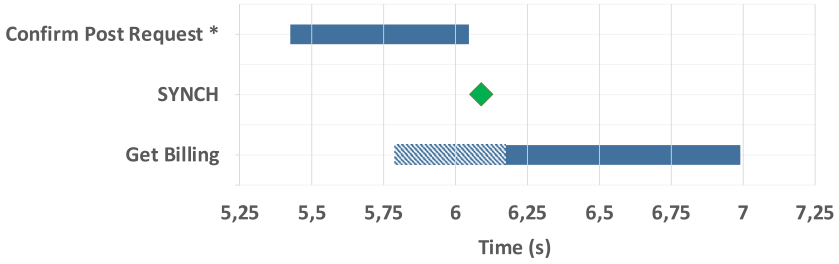

FIGURE 18 Details of the independent sequence execution

TABLE 2 Experiment results (data in ms)

	Tasks execution time			Coordination overhead			Adaptation overhead			Entity selection time		
Testing conditions	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**
1 User	6009	5602	6311	86.4	81	92	563	478	611	476	417	559
10 users	6022	5693	6304	86.7	82	91	568	484	609	474	425	541
50 users	6206	5923	6898	87.9	82	96	577	520	628	495	440	520
100 user	6704	6182	7348	92.1	88	99	599	521	673	501	462	546
200 users	7642	7002	8141	99.9	93	115	681	600	775	552	497	613
500 users	9198	8652	10036	113.8	95	149	802	732	904	671	585	729
1 User, 15 service instances	6042	5599	6284	86.5	81	90	573	481	623	478	409	510
1 User, 25 service instances	6055	5626	6341	87.5	83	92	585	502	668	481	413	543
1 User, 50 service instances	6078	5701	6400	88.3	83	92	597	511	695	488	426	544

- *Task Execution Time*—time between the first and the last logged timestamps. It is computed for each task as the difference between the timestamp of the request received by the initiating prosumer and the timestamp of (i) the received response (if the task is request-response) or (ii) the received request message by the receiving participant (otherwise);
- *Coordination Overhead*—time between the end of a task and the start of the following task, excluding the tasks that are initiated by the Mobile App or that require adaptation. It is computed as the difference between the timestamp for the end of the previous task and the timestamp of the request received by the initiating prosumer;
- *Adaptation Overhead*—time between the receiving of the request for the entity selection by the Context Manager and the start of the following task.

Moreover, we measured the *Entity Selection Time*, as the time needed to the Context Manager to select the adaptable entities and return the selection to the caCDs. It is computed as the difference between the two timestamps logged by the Context Manager. Note that this measure is already considered by the Adaptation overhead, but if observed individually it can give us information about the specific overhead introduced by Context Manager for the entities selection, in particular when considering sets of selectable service instances of different sizes.

Table 2 reports the results of our experiments. For each experimental setting, we report the average, minimum and maximum value for the total task execution time, total coordination overhead, total adaptation overhead and total entity execution time of each choreography run. The first six runs have been performed by increasing the level of concurrency by simulating a growing number of users, while the number of service instances into the Service Registry is fixed to the five instances that we deployed. In contrast, in the last three runs, we fixed the number of simulated users and we considered a growing number of service instances into the service registry. Each run has been repeated 20 times.

With a single user and five service instances, we measured an average total adaptation overhead of 563 ms. Of these, 476 ms are needed for the evaluation of the context and the selection of the adaptable entities, while the remaining 87 ms are spent for the execution of the adaptation performed by the caCDs. If considering the execution of a single adaptation, it requires an average of 141 ms, of which 119 ms needed for the entity selection. As a comparison, we measured that the task *Send Confirmation* lasts 164 ms, while the average execution time of a choreography task is 501 ms. The same can be said also with a higher degree of parallelism: with 500 instances, we measured a total overhead for the adaptation of 880ms (220 ms for a single selection), while the the average total time for the task execution is 9200 ms.

Figure 19 shows how the average times reported in Table 2 increase when the degree of parallelism increases. We observe that, when the number of concurrent requests grows, the mean total overhead for the adaptation has a decreasing weight with respect to the mean total

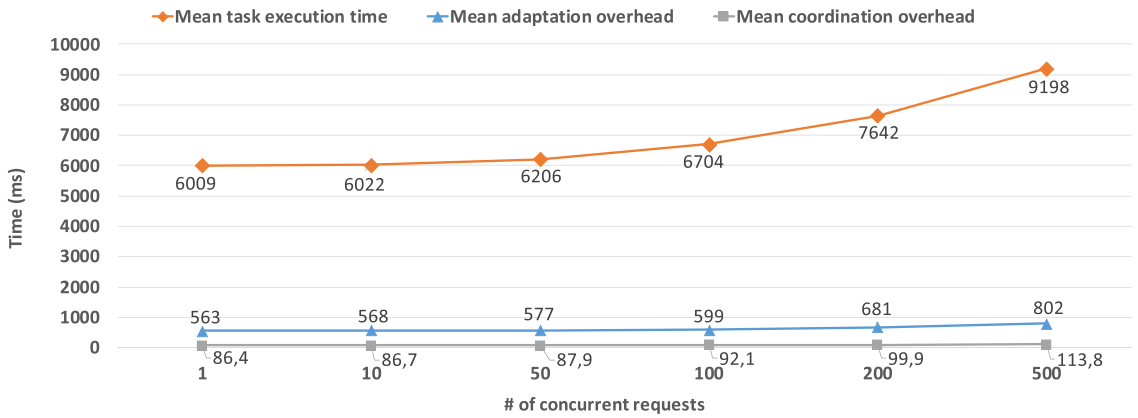

FIGURE 19 Trend of task execution time, adaptation overhead and coordination overhead at increasing number of requests

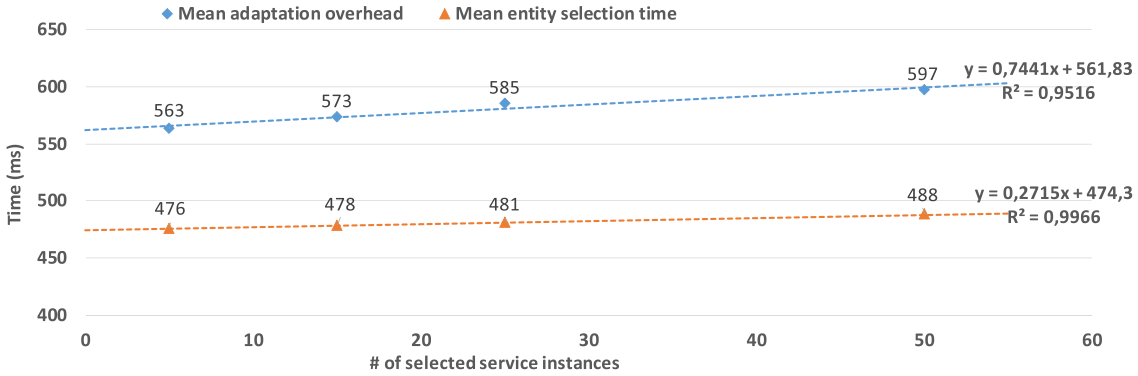

FIGURE 20 Trend of adaptation overhead and entities selection time at increasing number of service instances

execution time of the choreography tasks. In average, without concurrent users, the total time spent for the selection of the adaptable entities measures the 9.37% of the total task execution time; with 200 concurrent instances, it measures the 8.9%; with 500 concurrent instances, the 8.7%.

For what concerns the overhead for the coordination, we note that this overhead is even more negligible, since it represents only the 1,4% of the total task execution time without any concurrent instances, while it lowers to the 1,3% with 500 instances running concurrently.

A linear regression analysis performed using the mean values of the overhead shows that both adaptation and coordination overhead grow linearly with respect to the number of concurrent users. In particular, concerning the adaptation overhead, we observe that the value of the slope of the line is low (0.4933)—meaning that the growth is very smooth—and the coefficient of determination r^2 is very close to 1 (0.9843)—meaning that the linear growth provides a good approximation. The same holds for the coordination overhead: here, we observe that the slope is very low (0.0564); whereas, r^2 is 0.9872. The results concerning the coordination overhead support the findings that we obtained in previous work,[24] thus confirming the high scalability of the coordinated system when handling a high number of concurrent requests.

Finally, we observe how the adaptation overhead changes when the number of service instances that are selected increases. Figure 20 shows how the adaptation overhead and the entity selection time grow when the number of selected instances for the tasks with multi-instance participant increases. The linear regression highlights that both the adaptation overhead and the selection time have a linear growth. Interestingly, we can note that the latter grows with a lower slope than the whole adaptation overhead (0.2715 against 0.7741). This hints to the fact that the selection process is only marginally affected by the number of instances that can be selected. In contrast, the only instantiation process takes more time to be completed. In fact, by referring to Figure 10, we observe that, while the selection function is executed only once regardless of the number of service instances in the registry (interactions 1 and 2), the caCD asks the request message to the initiating participant of the task for each selected instance (interaction 3). This results in a higher number of interactions that are executed in parallel and that (slightly) increase the overhead. On the other hand, we can observe that the constant part of the overhead—represented by the y-intercept of the line—is mainly related to the selection time (474.3 ms, out of 561.83 ms of the whole adaptation constant part). This is due to the fact that, as explained, the selection process is always executed once even if no suitable service instances are found and selected: the context acquisition, the query on the

service registry, and the execution of the selection function represent computational steps that are always needed and executed only once, when adaptation is required.

7 | RELATED WORKS

The work presented in this paper is related to approaches for the choreography realization enforcement and the automated choreography realization, as well as context-aware and location based choreographed systems.

Choreography realizability and enforcement—Concerning the automated synthesis of choreographies and the choreography enforcement, our work leverages on the approaches presented in our previous works,[9,10,14] in which *Coordination Delegates* are introduced together with the automated synthesis process.

In the work of Gudemann et al,[25] choreography realizability is enforced through the generation of monitors, which act as local controllers interacting with their peers and the rest of the system in order to make peers respect the choreography specification. The notion of monitor is similar to our notion of CDs, since they coordinate the choreography by interacting with the other choreography participants.

Farah et al[26] address the realizability problem through an a-priori verification techniques using refinement and proof-based formal methods. Their approach is different from ours in that it defines and realizes a notion of peer projection that is correct by construction, avoiding the coordination from the outside and the introduction of additional software entities needed for enforcing realizability such as our CDs. Our approach, however, focuses on realizing a choreography by reusing third-party (and black-box) services rather than generating from scratch the correct peers. Only the application-specific components, together with the CDs, are generated.

Nguyen et al[27] introduce data support for analysing interaction-based service choreographies. This model uses data-aware interactions as the basic event. The authors propose a top-down development process for data-aware service choreographies by extracting a behavioral skeleton for each choreography participant through projection. In this way, developers have only to complete the skeletons with the business logic needed to realize the distributed application according to the choreography specification. Differently from our approach, which employs BPMN2 choreographies, their focus is on interaction-based service choreographies.

The ASTRO toolset[28] supports the automated composition of services. It aims to compose a service out of a business requirement and the description of available external services. 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 orchestration-based processes rather than decentralized choreography-based ones.

Context-aware adaptive systems—The concept of variability is widely used in association with Software Product Lines.[29-31] Several approaches have been introduced to facilitate the development of variable service architectures in the context of service-oriented SPLs.[32-36] However, they mainly focus on the architectural and development aspects of variability, without concerning the context-aware runtime adaptation of the service composition.

Yu et al[37] presented MoDAR, an approach to the development of dynamically adaptive service-based systems. The approach considers the development of two system models: a simplified business process (base model) and a variable model, which consists of a set of rules that drive the dynamic behavior of the system. A model-driven platform supports the semi-automatic transformation of the code into an executable system. Our approach is similar in that their base model is an underspecified model of the system, while (a part of) our variable part is defined through variants, and selection functions drive the dynamic behavior. However, we are more focussed on the context-awareness adaptation and we explicitly consider the context by defining its model.

Cubo et al[38] proposed an extension of DAMASCO, a framework for the service discovery and composition, by managing the variability of services and context during the service composition at runtime. They describe the service interfaces through a context profile that describes also the context information, and they use *feature models* in order to specify the service variability. The composition consider the context features in the model to find matching services with the adequate features, thus realizing adaptation. Differently from our approach, Cubo et al. consider the context changes during the composition through discovery means, while our adaptation is entirely performed at runtime with dynamic service selection according to the runtime conditions. Moreover, while they assume that context information is inferred by the client requests, we make use of acquisition functions in order to get context information also from outside the system.

Murguzur et al[39] introduce LateVa. In this framework, runtime variability is achieved through the definition of a base process model and process fragments. A fragment describes a single variant realization for each of the variation point defined into the base model. The latter is also annotated with endpoints from which context data is gathered. A variability model defines the variants and their context data mappings as feature models. At runtime, the fragment selector service searches for an available fragment in a model repository that can realize the required features. This approach tackles the adaptation challenge in a similar way with respect to our solution for the task-flow-level adaptation—through the definition of variants that are associated to variation points. However, we also explicitly consider the message and participant-level adaptation.

De Sanctis et al[40] presented an approach for designing and allowing the dynamic context-aware adaptation of service-based applications. Here, the adaptation since the design time and allows to divide the adaptation and the application logic in two separate models. This work

leverages their previous works,[41,42] in which abstract activities are defined at design time and are then refined at runtime when adaptation is performed through the continuous integration of new services. Unlike their approach, we avoid to model the adaptation process since the adaptation is performed by choosing adaptable entities among a well-defined set of candidates.

Riccobene et al[43] show SCA-ASM, a formal framework for modeling and executing service-oriented applications that are able to monitor and react to environmental changes. The approach exploits multi-agents that execute a distributed MAPE-K loop for monitoring the context, planning and execution adaptation both at architectural and at behavioral level. MAPE-K loops are used also in the work presented by Mongiello et al,[44] in which a service-oriented architecture is built at runtime by defining a metamodel based on knowledge graphs to model information about the environment around the user. Our approach differs from these since our adaptation strategy evaluates the context on demand without executing a control loop.

Calinescu and Rafiq[45] describe a method to automatically synthesize intelligent service proxies. They enable self-adaptation of a service-based system through the runtime selection of suitable participants in a workflow. Differently from our approach, they focus the selection to keep satisfying the high-level requirements of the system, dynamically selecting the services by means of online learning. In contrast, we focus the service selection on functional aspects rather than on non-functional requirements satisfaction.

De Prado et al[46] provide a scalable event-driven context-aware SOA architecture that exploits data obtained from IoT devices and offers context-aware REST services to the user. The approach described by the authors exploit an enterprise service bus that incorporate data coming from IoT devices and allows the communication between all involved agents. The context is managed by a context broker and a context DB, which keep the context information updated and provide the context information to the REST services. This approach focuses on the collection of context data by also using complex event processing, while we gather context information from business messages and exploit them on demand at runtime when selecting services.

8 | CONCLUDING DISCUSSION

In this paper we have presented an approach for the realization of context-aware choreographies. The proposed approach refines and extends the one envisioned in our previous work,[13] in which a generic context-awareness feature is realized only through an a priori specification of some well-defined alternative interactions. The context-aware adaptation is realized at three different levels: at the message level, at participant level, and at the task-flow level. We presented a novel notion of *Context-aware Coordination Delegates* that are able to realize the adaptation feature of the choreography while ensuring the coordination of services and their correct interaction. In order to represent the context, we introduced a metamodel which allows to define a context model that includes the description of the context characteristics, the sources of context data and the functions for the evaluation of the context. We have also introduced the notion of *context-carrying messages* in order to identify the business messages that carry useful context-related data that are exploited for the evaluation of the context and the dynamic selection of services.

We have presented a use case designed for the ConnectPA project, which leverages on the newly-introduced features in order to realize an e-government system for smart cities that exploits the composition of local-grade services owned by public authorities, and realizes a wider, dynamic, and interoperable system capable of easing and speeding up administrative processes.

The approach has been shown at work on the presented use case, and experiments have been performed in order to validate and evaluate the approach. We have set up an experimental environment in order to run the system and simulate the geographical distribution of the involved location-based services, and we stressed out the coordination and adaptation mechanism by simulating the interaction of multiple users that concurrently send requests to the choreography. The results show that our approach is able to effectively coordinate the services interaction when dealing with the *independent sequences* and with the independence of participant services (e.g., the Mobile App). On our use case the context, obtained through the evaluation of the content of the context-carrying messages, is properly addressed and the choreography behaves as a location-aware system able to provide to the user the access to the right local services according to the user's location. Moreover, in the scope of our use case, we have shown that the time overhead introduced for the selection of the adaptable entities is comparable to the execution time of a simple choreography task and it does not affect significantly the performances of the system. In our experimental deployment, the overhead for the adaptation grows less than the mean task execution time when the number of concurrent choreography instances increases, and most importantly it grows linearly with respect to both the number of users and number of service instances.

The results of the experimentation obtained in this paper are considered as a first-sight evaluation of the proposed approach, and they have to be observed more in deep and in a larger variety of cases in order to be widely accepted. However, they suggest that the approach proposed in this paper is suitable for the ConnectPA use case, which is being realized in synergy with the industrial partners involved in the project.

Limitations of the approach—For sake of simplicity, we presented the participant-level adaptation leveraging on the assumption that all the instances of a multi-instance participant service share the same interface and the same interaction protocol. Although this holds in our use case, often different service instances can offer the same functionalities while exposing different interfaces. This assumption allowed us to focus on the context-aware adaptation rather than on other issues. However, in order to deal with interface incompatibilities, we can introduce service

Adapters as already shown in previous works.[47] Adapters are software entities that are automatically synthesized and bridge the gap between the interfaces of the selected services and their abstract description in the choreography model. When a service instance with an incompatible interface is added to the Service Registry, an adapter can be synthesized in order to be used when the instance is selected at runtime.

The service selection is a dynamic adaptation mechanism that does not rely on pre-defined alternatives. Services can be added or removed at runtime from the service registry without requiring any reconfiguration or re-deploy. In fact, the service selection function selects the services that are suitable according to the context (of both system and entities) without knowing a-priori the set of candidate services instances. The instances are selected among those that are in the registry right when the selection is performed. Context acquisition functions can also provide information about the current service availability and/or their QoS levels in order allow the adaptation also according to the dynamic conditions of the participant services. In contrast, our task-level and task-flow level adaptation (i.e., choreography variants) relies on a set of well-defined alternatives that are defined since the early stages of the choreography modeling. The selection function selects among a set of known alternatives, by returning the name of the selected variant to the caCDs. It is clear that, in this case, adding a new variant at runtime requires a reconfiguration of the system. However, we need to only update the coordination protocol of the caCDs that handle the new variant, in order to suitably coordinate the participant services, and—if needed—the selection function in the Context Manager so as to consider the updated set of adaptable entities (i.e., variants among which the selection has to be performed).

Future work—Future work concerns further experimentations of the approach in relation to different scenarios, by considering both different deployments of the choreography and different use cases. The experimentation on different use cases will allow us to evaluate the approach against systems that have stringent performances constraints or that need a more complex process for the evaluation of the context. Moreover, this will allow us to better evaluate the impact of the assumptions that have been made and the limitations of the approach.

Then, as anticipated in Section 4.3, we aim to overcome the assumptions made concerning (i) possible context changes happening during when the selected services are serving a request and (ii) the stateless vs. stateful nature of the services. We plan to exploit a smarter form of quiescence (e.g., inspired by the tranquility state[48]) to realize a mechanism that, in the case of stateless services, keeps the current input messages being processed by the service instances to be substituted, rollbacks the possible modification being made, activate the new instance and pass the kept input parameters to it. In the case of stateful services with internal state, use ad-hoc methods for saving the (current) internal state of the service instance to be substituted and restoring it into the new instance to be engaged. There is no need to rollback if the quiescence/tranquility state is properly detected.

Finally, we plan to enhance the architecture of the Context Manager so as to distribute its functionalities, by providing to each caCD the capability of autonomously evaluate the context and select the adaptable entities, rather than having a single service that accomplishes these functions.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Gianluca Filippone https://orcid.org/0000-0001-8908-6960
Marco Autili https://orcid.org/0000-0001-5951-1567
Massimo Tivoli https://orcid.org/0000-0001-9290-1997

REFERENCES

1. Unver MB. Turning the crossroad for a connected world: reshaping the European prospect for the Internet of Things. *Int J Law Inform Technol*. 2018; 26(2):93-118.
2. Lee HJ, Kim M. The Internet of Things in a smart connected world. In: Sen J, ed. *Internet of things - technology, applications and standardization*. London, UK: IntechOpen; 2018:91-104.
3. Akyildiz IF, Kak A. The Internet of Space Things/CubeSats: a ubiquitous cyber-physical system for the connected world. *Elsevier J Comput Netw*. 2019; 150:134-149.
4. Patrono L, Atzori L, Šolić P, Mongiello M, Almeida A. Challenges to be addressed to realize Internet of Things solutions for smart environments. *Future Gen Comput Syst*. 2020;111:873-878.
5. Hernández-Muñoz JM, Vercher JB, Muñoz L, et al. Smart cities at the forefront of the future internet. In: Domingue J, Galis A, Gavras A, et al., eds. *The future internet*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011:447-462.
6. Purohit L, Kumar S. Web services in the Internet of Things and smart cities: a case study on classification techniques. *IEEE Consumer Electron Mag*. 2019;8(2):39-43.
7. Palattella MR, Dohler M, Grieco LA, et al. Internet of Things in the 5G era: enablers, architecture, and business models. *IEEE J Sel Areas Commun*. 2016; 34(3):510-527.
8. Chettri L, Bera R. A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. *IEEE Int Things J*. 2020;7(1):16-32.
9. Autili M, Salle AD, Gallo F, Pompilio C, Tivoli M. Chorevolution: service choreography in practice. *Sci Comput Programm*. 2020;197:102498.
10. Autili M, Inverardi P, Tivoli M. Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates. *Sci Comput Programm*. 2018;160:3-29.

11. Corradini F, Fornari F, Polini A, Re B, Tiezzi F. A formal approach to modeling and verification of business process collaborations. *Sci Comput Programm*. 2018;166:35-70.
12. Basu S, Bultan T. Automated choreography repair. In: Stevens P, Wąsowski A, eds. *Fundamental approaches to software engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2016:13-30.
13. Filippone G, Autili M, Tivoli M. Towards the synthesis of context-aware choreographies. In: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE; 2020:197-200.
14. Autili M, Di Salle A, Gallo F, Pompilio C, Tivoli M. Aiding the realization of service-oriented distributed systems. In: Proceedings of the 34th ACM/-SIGAPP Symposium on Applied Computing, SAC '19. Association for Computing Machinery; 2019; New York, NY, USA:1701-1710.
15. Dey AK. Understanding and using context. *Personal Ubiquitous Comput*. 2001;5(1):4-7.
16. Schilit B, Adams N, Want R. Context-aware computing applications. In: 1994 First Workshop on Mobile Computing Systems and Applications; 1994: 85-90.
17. Zainol Z, Nakata K. Generic context ontology modelling: a review and framework. In: 2010 2nd International Conference on Computer Technology and Development; 2010:126-130.
18. Bauer C, Novotny A. A consolidated view of context for intelligent systems. *J Ambient Intell Smart Environ*. 2017;9(4):377-393.
19. Satyanarayanan M. Pervasive computing: vision and challenges. *IEEE Wirel Commun*. 2001;8(4):10-17. https://doi.org/10.1109/98.943998
20. Bettini C, Brdiczka O, Henricksen K, et al. A survey of context modelling and reasoning techniques. *Pervasive Mob Comput*. 2010;6(2):161-180.
21. Han L, Salomaa JP, Ma J, Yu K. Research on context-aware mobile computing. In: 22nd International Conference on Advanced Information Networking and Applications, AINA 2008, Workshops Proceedings, Ginowan, Okinawa, Japan. IEEE Computer Society; 2008:24-30.
22. Nitto ED, Ghezzi C, Metzger A, Papazoglou MP, Pohl K. A journey to highly dynamic, self-adaptive service-based applications. *Autom Softw Eng*. 2008; 15(3-4):313-341.
23. Leite LAF, Oliva GA, Nogueira GM, Gerosa MA, Kon F, Milojicic DS. A systematic literature review of service choreography adaptation. *Serv Oriented Comput Appl*. 2013;7(3):199-216.
24. Autili M, Perucci A, Leite L, Tivoli M, Kon F, Di Salle A. Highly collaborative distributed systems: synthesis and enactment at work. *Concurrency Comput Pract Exper*. 2021;33(6):e6039.
25. Güdemann M, Salaün G, Ouederni M. Counterexample guided synthesis of monitors for realizability enforcement. In: Chakraborty S, Mukund M, eds. *Automated technology for verification and analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012:238-253.
26. Farah Z, Aït-Ameur Y, Ouederni M, Tari K. A correct-by-construction model for asynchronously communicating systems. *Int J Softw Tools Technol Transf*. 2017;19(4):465-485.
27. Nguyen HN, Poizat P, Zaidi F. Automatic skeleton generation for data-aware service choreographies. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE); 2013:320-329.
28. Trainotti M, Pistore M, Calabrese G, et al. Astro: Supporting composition and execution of web services. In: Benatallah B, Casati F, Traverso P, eds. *Service-oriented computing - ICSOC 2005*: Springer Berlin Heidelberg; 2005:495-501.
29. Babar MA, Chen L, Shull F. Managing variability in software product lines. *IEEE Softw*. 2010;27(3):89-91, 94.
30. Webber DL, Gomaa H. Modeling variability in software product lines with the variation point model. *Sci Comput Programm*. 2004;53(3):305-331.
31. Bastida L, Nieto FJ, Tola R. Context-aware service composition: a methodology and a case study. *Proceedings of the 2nd International Workshop on Systems Development in soa Environments*. New York, NY, USA: Association for Computing Machinery; 2008:19-24.
32. Boffoli N, Cimitile M, Maggi FM, Visaggio G. Managing SOA system variation through business process lines and process oriented development. In: Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL); 2009:61-68.
33. Mohabbati B, Hatala M, Gašević D, Asadi M, Bošković M. Development and configuration of service-oriented systems families. In: Proceedings of the 2011 ACM Symposium on Applied Computing. Association for Computing Machinery; 2011:1606-1613.
34. Razavian M, Khosravi R. Modeling variability in the component and connector view of architecture using UML. In: 2008 IEEE/ACS International Conference on Computer Systems and applications; 2008:801-809.
35. Sun H, Lutz RR, Basu S. Product-line-based requirements customization for web service compositions. *Proceedings of the 13th International Software Product Line Conference*. USA: Carnegie Mellon University; 2009:141-150.
36. Topaloglu Y, Capilla R. Modeling the variability of web services from a pattern point of view. *Web services*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004:128-138.
37. Yu J, Sheng QZ, Swee JKY. Model-driven development of adaptive service-based systems with aspects and rules. In: Chen L, Triantafillou P, Suel T, eds. *Web Information Systems Engineering—Wise 2010*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010:548-563.
38. Cubo J, Gamez N, Fuentes L, Pimentel E. Composition and self-adaptation of service-based systems with feature models. In: Favaro J, Morisio M, eds. *Safe and secure software reuse*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013:326-342.
39. Murguzur A, Trujillo S, Truong H-L, Dustdar S, Ortiz O, Sagardui G. Run-time variability for context-aware smart workflows. *IEEE Softw*. 2015;32(3): 52-60.
40. De Sanctis M, Bucchiarone A, Marconi A. Dynamic adaptation of service-based applications: a design for adaptation approach. *J Internet Serv Appl*. 2020;11(1):2.
41. Bucchiarone A, De Sanctis M, Marconi A, Pistore M, Traverso P. Design for adaptation of distributed service-based systems. In: Barros A, Grigori D, Narendra NC, Dam HK, eds. *Service-oriented computing*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2015:383-393.
42. Bucchiarone A, De Sanctis M, Marconi A, Pistore M, Traverso P. Incremental composition for adaptive by-design service based systems. In: Reiff-Marganiec S, ed. *IEEE International Conference on Web Services, ICWS 2016, San Francisco, CA, USA*: IEEE Computer Society; 2016:236-243.
43. Riccobene E, Scandurra P. Formal modeling self-adaptive service-oriented applications. In: Wainwright RL, Corchado JM, Bechini A, Hong J, eds. *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain*: ACM; 2015:1704-1710.
44. Mongiello M, Noia TD, Nocera F, Sciascio ED. Case-based reasoning and knowledge-graph based metamodel for runtime adaptive architectural modeling. In: Ossowski S, ed. *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy*: ACM; 2016:1323-1328.
45. Calinescu R, Rafiq Y. Using intelligent proxies to develop self-adaptive service-based systems. In: Seventh International Symposium on Theoretical Aspects of Software Engineering, TASE 2013. IEEE Computer Society; 2013; Birmingham, UK:131-134.
46. García De Prado A, Ortiz G, Boubeta-Puig J. Cared-SOA: a context-aware event-driven service-oriented architecture. *IEEE Access*. 2017;5:4646-4663.

47. Autili M, Di Salle A, Gallo F, Pompilio C, Tivoli M. Model-driven adaptation of service choreographies. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18. Association for Computing Machinery; 2018; New York, NY, USA:1441-1450.
48. Vandewoude Y, Ebraert P, Berbers Y, D'Hondt T. An alternative to quiescence: tranquility. In: 2006 22nd IEEE International Conference on Software Maintenance; 2006:73-82.

How to cite this article: Filippone G, Autili M, Tivoli M. Synthesis of context-aware business-to-business processes for location-based services through choreographies. *J Softw Evol Proc*. 2021;e2416. doi:10.1002/smr.2416