Dealing with the evolution of event-based choreographies of BPMN fragments

Jesús Ortiz

Universitat Politècnica de València, PROS Research Centre, Valencia, Spain

Abstract

Business processes (BPs) are commonly used by organizations to describe their goals. However, the existent decentralization found in many organizations forces them to build such BPs by coordinating distributed and fragmented BPs. Within this context, microservices arise as a very interesting and convenient way to address the implementation of such processes due to their low coupling character. In this case, the coordination of such fragmented BPs is usually achieved by means of event-based choreographies. Though, one of the main challenges to be faced by choreographies is their evolution due to the complexity that introduces the need of integrating changes among autonomous and independent partners. To this end, this thesis work faces the challenge of evolving a microservice composition that is globally defined in a BPMN collaboration diagram but executed through a choreography of BPMN fragments. To do so, it presents a characterization of the changes that can be performed from the local perspective of a microservice and the creation of a catalogue of adaptation rules that can be applied to the rest of microservices to maintain the functional integrity of the composition. In addition, a microservice architecture supporting this composition approach is proposed, which includes a MAPE-K loop component for the selection of the adaptation rules when a change is introduced. Finally, a protocol is defined for the application of the adaptation rules in order to ensure the propagation of the changes and the adaptations rules to the rest of participants.

Keywords

Microservice, composition, BPMN, MAPE-K, evolution.

1. Introduction

Business processes (BPs) are the key instrument to organizing and understanding the interrelationships of the different activities required to produce an outcome to the market [1]. However, when these activities are performed in a decentralized way, e.g., by different departments within the same organization, the decoupling characteristic of microservices makes them a very interesting and convenient way to implement such processes. Microservice architectures [2] propose the decomposition of applications into small independent building blocks (the microservices) that focus on single business capabilities.

Therefore, microservices need to be composed to support the BPs of organizations. To this end, to keep a lower coupling and independence among microservices for deployment and evolution, these compositions are usually implemented by means of event-based choreographies. However, choreographies split the control flow of the composition among the different participant microservices, making it hard to analyze and understand when requirement changes. To improve this problem, [5] proposed an approach based on the choreography of BPMN fragments. In this approach, business process engineers create a microservice composition through a BPMN collaboration diagram. Then, this diagram is split into BPMN fragments which are executed through an event-based choreography.

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal

jortiz@pros.upv.es (J. Ortiz)

0000-0002-9352-1045 (J. Ortiz)

This composition approach is supported by a microservice architecture to achieve that both descriptions of a composition, the *global composition view* and the *split view*, coexist in the same system. However, this solution introduces a new challenge to be faced: how to evolve a microservice composition that is globally defined in a BPMN collaboration diagram but executed through a choreography of BPMN fragments. This work does not adequately support the evolution from the local perspective of a BPMN fragment since changes that have an impact in other microservices are not considered.

To this end, in this thesis work we face this problem by characterizing the local changes that can be introduced from the local perspective of a BPMN fragment and defining a catalogue of adaptation rules that are applied when a change introduces inconsistencies in the composition, in order to maintain the functional integrity of the composition. In addition, a microservice that implements a MAPE-K loop is introduced in the presented architecture that supports this composition approach. This microservice automatically selects the adaptations rules that must be applied from the catalogue. Additionally, a protocol to facilitate the propagation of local changes and adaptation rules to the rest of microservices is also presented. The contributions of this work are focused on improving the problem of automating the redesign of the processes of a microservice composition, in this case, implemented as a choreography of BPMN fragments, which currently remains as a manual and cognitively demoing task, making it time-consuming, labor-intensive and error-prone [4].

The rest of the paper is organized as follows: Section 2 presents an overview of the work this thesis relies on. Then, section 3 presents the catalogue of adaptation rules required to keep the functional integrity of the choreography, and section 4 the microservice architecture designed to support the evolution challenge previously identified. Then, section 5 analyses the related work, and finally, conclusions are commented on in Section 6.

2. Previous work and contribution

This section presents an overview of the approach proposed in [5] to create a microservice composition as an event-based choreography of BPMN fragments. This approach proposes to create a microservice composition in two main steps (see Figure 1): (1) business process engineers create the *global composition view* of the composition in a BPMN collaboration diagram following an orchestration approach, and (2) the BPMN collaboration diagram is split into BPMN fragments that are deployed into separated microservice and executed through an event-based choreography. Each BPMN fragment is sent to its corresponding microservice and managed autonomously by its microservice development team.

As a representative example, a scenario based on the e-commerce domain is defined, which describes the process of placing an order in an online shop. The different tasks that make up this process are distributed by responsibilities in four different lanes: *Customers*, *Inventory*, *Payment* and *Shipment* (see top of Figure 1).

After creating the global composition view of the composition, the BPMN collaboration diagram is split by responsibilities into independent BPMN fragments that will be managed by different microservices (see bottom of Figure 1). This split is performed automatically by a tool developed in [5] where each BPMN fragment is created as a pool with the tasks that are defined in the corresponding lane and a list of Catch/Throwing Events that are automatically added to support the event-based choreography. The microservices managing each fragment are endowed with a process engine that oversees the execution of their respective BPMN fragment to execute (1) the tasks defined in the pool (what we call *functional requirements*), and (2) the Catch/Throwing Events to either receive or publish asynchronous events in a communication bus to support the collaboration with the rest of participants (what we call *coordination requirements*). Thus, the microservice composition is executed by means of an event-based choreography of BPMN fragments in which microservice wait for specific events to execute their corresponding piece of work.

This solution introduces two main benefits: First, it facilitates the analysis of the control flow since a BPMN diagram that represents the entire composition (e.g., the *global composition view*) is available. Second, it provides a high level of decoupling among the microservices that participate in the composition. However, this solution introduces a new challenge to be faced: how to evolve a microservice composition that is globally defined in a BPMN collaboration diagram but executed

through a choreography of BPMN fragments. This work allows the introduction of changes from two perspectives: from a global perspective, i.e., modifying the BPMN diagram that represents the *global composition view* (top-down evolution); and from a local perspective, i.e., modifying the BPMN fragment of one microservice (bottom-up evolution). While the top-down evolution is natively supported by the tool presented in [5], the bottom-up evolution is limited to some specific changes, specifically to the changes that can be managed in isolation, without affecting other microservices.

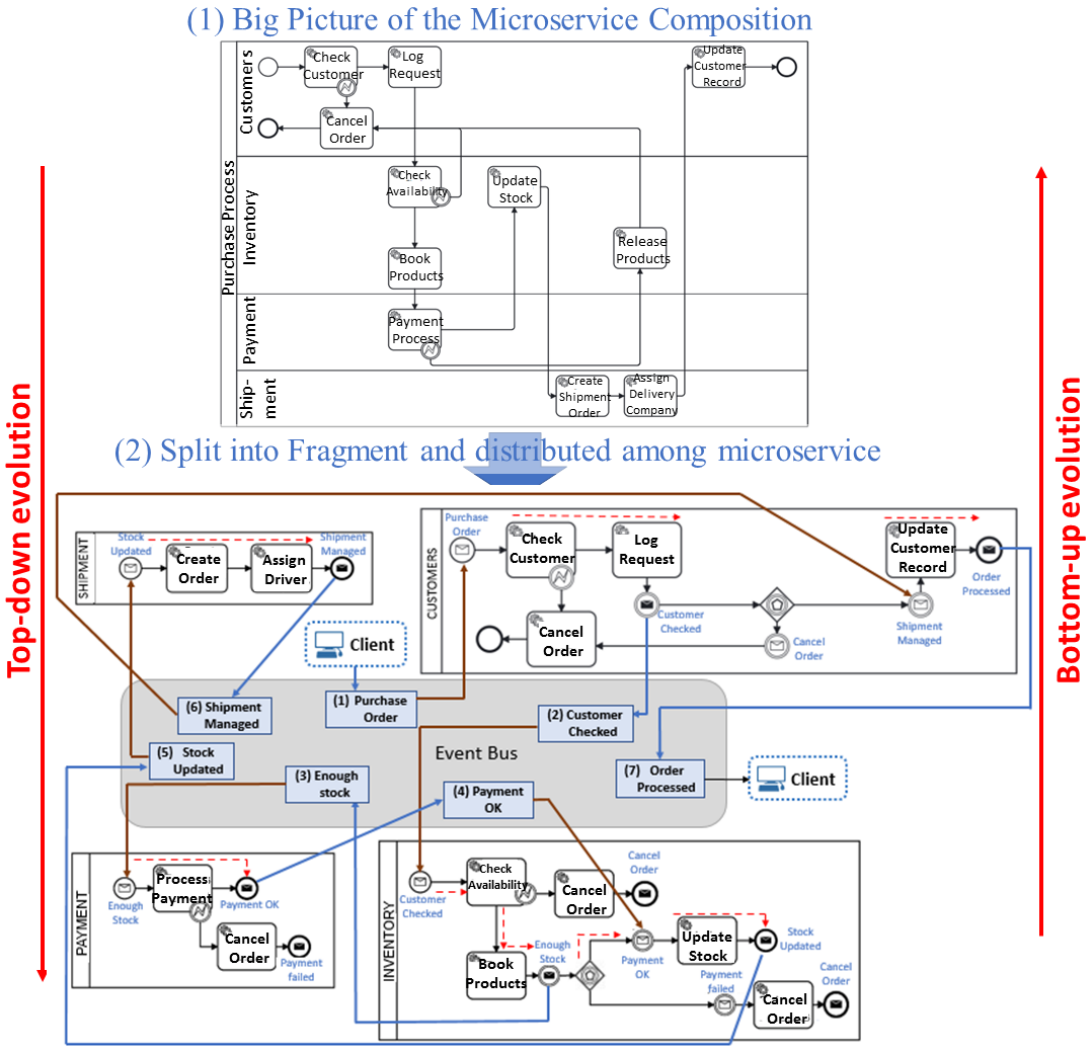

Figure 1: A microservice composition based on BPMN fragments

To address this challenge, in this thesis we face the evolution of a microservice composition from a bottom-up perspective when a participant introduces a local change in its individual BPMN fragment that affects other microservices. When changes are introduced in this scenario, additional aspects must be guaranteed due to the complexity introduced by the interaction of autonomous and independent partners. Therefore, adaptations to maintain the integrity of the choreography should be suggested to the affected partners. Additionally, following the proposed microservice composition approach, a change introduced from the local perspective of a microservice needs to be integrated with both, the BPMN fragments of the rest of the microservices and the BPMN collaboration diagram that represents the *global composition view*. Allowing local changes in a microservice reinforces the independence among development teams that is demanded by this type of architecture. Note that a change may affect a *functional requirement* (e.g., modifying a BPMN task) or a *coordination requirement* (e.g., modifying a Catch/Throwing Event). While local changes in *functional requirements* only affect the internal behavior of a microservice and do not require the adaptations of the rest of participants, local changes in *coordination requirements* can have an impact on the composition, eventually causing its failure (e.g., affecting the participation of the rest of microservices). For instance (see bottom of Figure 1), if

the *Customers* microservice deletes the Throwing Event that sends the message *Customer Checked*, the *Inventory* microservice, which is waiting for it, will never start, and the microservice composition will never continue. Therefore, changes in these types of requirements imply coordinated actions in two or more microservice in such a way a correct communication between microservice is ensured.

Consequently, this thesis work faces the challenge of supporting the bottom-up evolution of a microservice composition, focusing on the *coordination requirements* found in distributed compositions. To do so, the contributions of this work are:

1. The characterization of the changes that can be performed from the local perspective of a microservice and the creation of a catalogue of adaptation rules that can be applied to the rest of microservices in order to maintain the functional integrity of the composition.
2. The proposal of an extended microservice architecture that integrates a MAPE-K loop for the selection of adaptation rules when local changes are introduced.
3. The definition of a protocol for the application of the adaptation rules that allows the acceptance of developers, when needed, to guarantee the propagation of the changes and the adaptations to the rest of the participants of the composition.

3. Catalogue of adaptation rules

A catalogue of adaptation rules that describe how a local change must be managed to maintain, when possible, the functional integrity of the choreography, has been defined. To guarantee the functional integrity, it must be ensured that all microservices must be able to participate in the choreography and therefore, there must be at least one Throwing Event that sends a message to be received by a Catching Event. However, the rules that are included in the catalogue do not ensure properties such as deadlock-free and fault-tolerance in a composition. These problems have been extensively researched by the BPM community [16, 17] and to face them, the change patterns identified in [3] are proposed, in order to ensure the correctness of the composition after the change and the adaptation rule are applied.

A total amount of 14 adaptations rules have been defined and they have been collected in the following catalogue of adaptation rules [6]. The rules can be defined as a set of modifications actions that are applied to the BPMN fragments affected by the introduced change, i.e., those BPMN fragments that, due to the local change, can no longer participate in the composition. Therefore, the adaptation rules apply the required modifications to maintain the participation of the affected BPMN fragments in the choreography. Based on [7], every add, delete, and update modification that can be applied in a *coordination requirement* has been identified. 6 out 14 rules are defined to support delete changes, 6 out of 14 rules are defined to support update changes, and finally, the 2 remaining rules are defined to support add changes. Only these types of modifications have been identified as they are challenging by themselves, and any further change can be written as a combination of them. All modifications identified have been applied in four different case studies[2], and the adaptation actions required to maintain the functional integrity of the composition have been analyzed. This was done following an iterative and incremental process [18] in such a way the adaptation rules were progressively developed and tested in the case studies, refining previous definitions when some errors or objections were detected.

In this paper, due to space limitations, only one of the rules of the catalogue is presented, specifically Adaptation Rule #1. This rule faces the removal of a Throwing Event in a BPMN fragment. This type of change modifies a BPMN fragment by removing a BPMN element that is used to send a message to inform other microservices that a piece of work has been done. Thus, it affects all the microservices that include a Catching Event to receive the deleted message, which will never start or continue since their execution depends on the triggering of such message. Therefore, to support this local change, Adaptation Rule #1 adapts the affected microservices to start listening to the message triggered just before the deleted one.

As a representative example, the example presented in Section 2 is used to show how Adaptation Rule #1 is applied (see Figure 2). If the Throwing Event *Customer Checked* of the *Customers*

[2] The considered case studies can be found in: https://github.com/microserviceresearch/ml-microservice-composition-evolution/tree/main/CaseStudies

microservice is removed, the *Inventory* microservice, which is waiting for it, will never start. To allow the *Inventory* microservice to complete its tasks, it is modified to wait for the message previously caught by the modified fragment, i.e., to wait for the *Process Purchase Order* message instead. This adaptation of the affected local fragment maintains the functional integrity of the choreography (i.e., all the tasks executed before the local change are executed afterwards). However, two microservices that initially worked sequentially (e.g., first *Customers* checks the customer information, and then *Inventory* checks the availability of the purchased products) are now executed in parallel (e.g., after the adaptation, the start of the *Customers* microservice and the *Inventory* microservice are executed when the client sends the message *Process Purchase Order*).

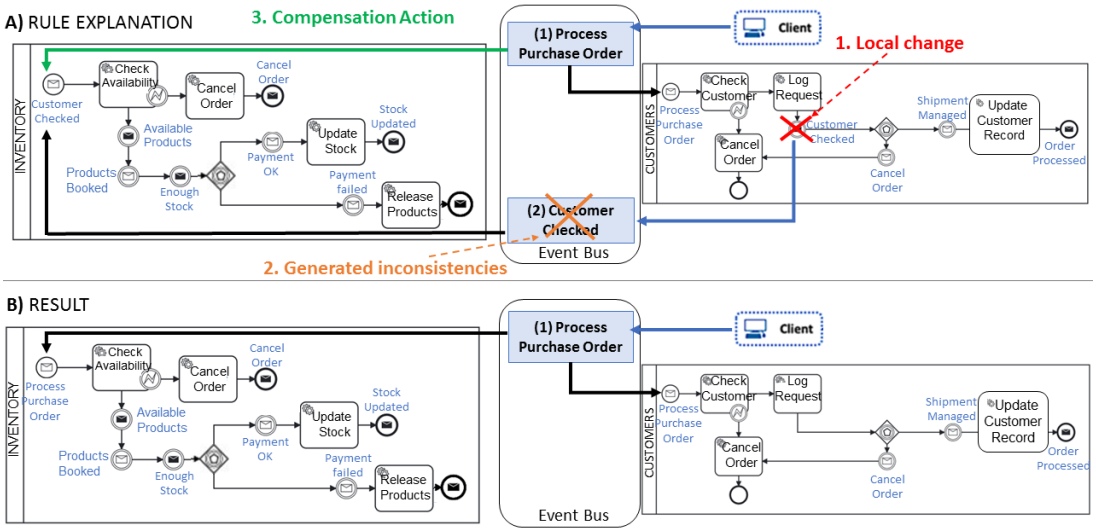

Figure 2: Example of Adaptation Rule #1

4. Supporting microservice architecture

In this section, an extension to [5], a microservice architecture that supports a microservice composition based on the choreography of BPMN fragments, is described (see Figure 3). The objective of this extended architecture is to support not just the development and deployment of a microservice composition based on BPMN fragments, but also its further evolution. For this purpose, new components are introduced into the architecture to automate, as much as possible, the application of adaptation rules to maintain the integrity of the composition when a change in a *coordination requirement* is done from the local perspective of one microservice.

The microservice architecture has been extended by introducing a MAPE-K control loop component, which is typically used to manage the adaptations of autonomic systems [19]. A MAPE-K control loop consists of four phases: the 1) **M**onitoring phase; the 2) **A**nalysis phase; the 3) **P**lanning phase; and the 4) **E**xecution phase. In addition, the MAPE-K control loop includes a **K**nowledge Base that stores properties to describe the past and present state of the system and its environment [20].

Initially, the architecture presented in [5] was designed with the following three components: (1) the business microservices that participate in the composition. Following the example of Section 2, we have the *Customers*, *Inventory*, *Payment* and *Shipment* microservices. (2) the *Global Manager* microservice that oversees the management of the BPMN collaboration diagram that represents the *global composition view*, and (3) an event bus that supports the exchange of messages among microservices at execution time. To support the evolution of the composition, the architecture has been revisited as follows: (4) the *MAPE-K Controller* microservice that controls the three first phases of the MAPE-K loop has been introduced, (5) the *Global Manager* microservice has been endowed with the adaptation rules and a new component that oversees the execution phase of the MAPE-K loop; and (6) the *Knowledge Base* component to register the local changes that occur in the system has also been included.

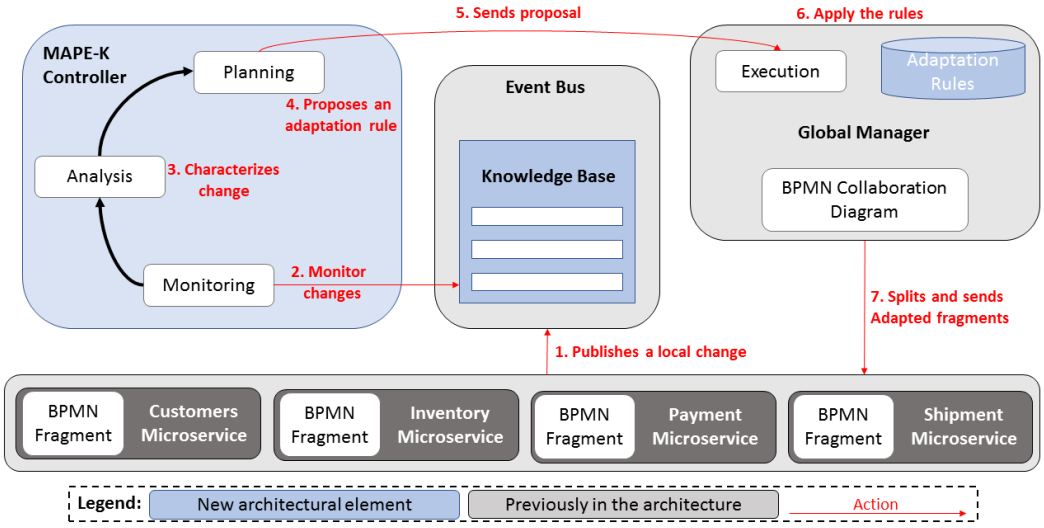

Figure 3: Representation of the architecture

The *MAPE-K Controller* is the component that oversees the first three phases of the MAPE-K loop, i.e., the Monitoring phase, the Analysis phase, and the Planning phase. The *Global Manager* is in charge of the last phase of the MAPE-K loop (i.e., the Execution phase), and is endowed with the catalogue of adaptation rules. Each rule is considered as an endogenous model transformation [21], i.e., a transformation between two models (the original BPMN fragment and the adapted one) expressed in the same language. In this work, we have used a direct manipulation approach based on the Java BPMN parser provided by the Camunda platform. We have selected this option since it is supported by other Java tools that facilitate the integration of the adaptation rules with the previous microservice architecture. Finally, the *Knowledge Base* component is defined to represent and store the local changes that occur in the system to be able to react to them. Therefore, the *Knowledge Base* can be considered as a collection of logs where the microservices indicate which element within their BPMN fragment has been changed.

To properly understand the MAPE-K loop implemented in the architecture, additional details are given first about the Knowledge Base and then about the MAPE-K phases:

Knowledge Base: In this work, we want to evolve a microservice composition that is implemented as an event-based choreography of BPMN fragments when a local change is done from the local perspective of one microservice. Thus, each time a microservice performs a local change in its BPMN fragment, the change is published in the event bus (Step 1 in Figure 3). Then, the *Knowledge Base* stores the published change indicating which element within the modified BPMN fragment has been changed.

Monitoring phase: This phase identifies when a published local change can introduce inconsistencies (Step 2 in Figure 3). In this work, an inconsistency occurs when a change in a *coordination requirement* is made, since it can affect the participation of the microservices in the choreography, eventually causing its failure. Changes in *functional requirements* can be managed internally without affecting the rest of the partners, and therefore, they do not require the application of any adaptation rule. The changes that can produce inconsistencies in *coordination requirements* are analyzed in detail in [6].

Analysis phase: This phase collects and analyses all the necessary information to characterize a local change in detail. The information required to perform such characterization is collected from the local change registered in the *Knowledge Base* (Step 3 in Figure 3). This characterization of the local change is done to be interpreted by a machine learning algorithm used in the Planning phase. When all this information is collected, it is encoded as a feature vector [8]. For simplification purpose, only five of the features that are collected are presented: (F1) the type of modified element (Throwing Event or Catching Event); (F2) the type of change that has been done (we only consider three type of changes, delete, update, or add); (F3) whether or not the change results in the publication of a new message in the choreography; (F4) whether or not a Throwing Event of another BPMN fragment is affected; and (F5) whether or not a Catching Event of another BPMN fragment is affected. As representative

example, Table 1 represent a partial characterization of the local change presented in Figure 2: the modified element (F1) is a Throwing Event (represented by the value 1), the action done (F2) is a deletion (represented by the value of 0); the change (F3) does not result in the publication of a new message since is a deletion (represented by the value of 0); a BPMN Throwing Event (F4) is not affected (represented by the value of 0), and a BPMN Catching Event (F5) is affected (represented by the value of 1).

Table 1
A partial feature vector characterizing the delete change illustrated in Figure 2

F1	F2	F3	F4	F5	…
1	0	0	0	1	

Planning phase: Once a local change has been characterized in a feature vector, the planning phase must select one of the adaptation rules contained in the catalogue presented above to solve the inconsistencies created by the change (Step 4 in Figure 3). To do so, we propose using an algorithm based on machine learning techniques, which provides us with several benefits. Note that by manually implementing an algorithm to predict an adaptation rule, we need to code a vast amount of complex condition, which is a time-consuming and error-prone task. In addition, this makes its further redesign difficult. Using a machine learning algorithm, this process is automatically done by a prediction model that just needs to be trained with data. Also, we can re-train the prediction model or change the technique used to predict either to improve the predictor's performance or to add more adaptation rules in the future. Thus, in the planning phase, the feature vector that characterizes the local change is processed by a machine learning algorithm that selects an adaptation rule to face the change.

Execution phase: Once the machine learning algorithm has selected an adaptation rule to be applied, the *Global Manager* microservice is informed (Step 5 in Figure 3) to apply it in the BPMN collaboration diagram that represents the *global composition view* (Step 6). To do so, we apply a protocol (presented in the next section) that classifies the adaptation rules into *automatic adaptation* or *adaptation with acceptance*. Depending on this classification, the *Global Manager* applies the rule either automatically or previous human acceptance (Step 7 in Figure 3).

5. Proposed evolution protocol

This section explains the proposed evolution protocol to facilitate the propagation of changes when adaptation rules must be applied. Note that the impact of change propagation on running instances is not addressed by the proposed protocol. The adaptations applied only affect the new instances created after it. This decision is technologically supported by the default behavior of the BPMN engine that is used to execute the microservice composition, which provides a versioning strategy to evolve the process definition without affected running instances.

Broadly speaking, to define a protocol we need to describe the participants, the actions each participant does within the protocol and the messages they exchange [9]. The participants involved in the proposed protocol can be either software participants, represented by microservices which perform actions automatically (i.e., the locally modified microservices and the affected ones), or human participants, represented by process stakeholders who participate in the design and the decision-making process required to apply the proposed evolution protocol (i.e., the developers of the microservices and the BP engineer responsible for the *Global Manager* microservice). Regarding the actions and the interchange messages considered by the protocol are shown in Figure 4. When an adaptation rule has been selected, this can be classified as *automatic adaptation* or *adaptation with acceptance*.

On the one hand, an adaptation rule is classified as *automatic adaptation* when the adaptation required to maintain the integrity of the choreography can be automatically applied in the BPMN collaboration diagram that represents the *global composition view* since the *functional* and *coordination requirements* are both maintained. Therefore, no human participation is required. Following the protocol presented in Figure 4, when an adaptation rule is selected and classified as *automatic adaptation*, the rule is applied on the BPMN collaboration diagram that represents the *global composition view*, stored in the *Global Manager*, to afterwards, re-split the BPMN collaboration

diagram into BPMN fragments, and send the adapted fragments to the corresponding microservices to be implemented. Finally, a commit message is sent to the modified microservice to inform that the adaptation has been applied and consequently, it can implement the introduced change in its BPMN fragment.

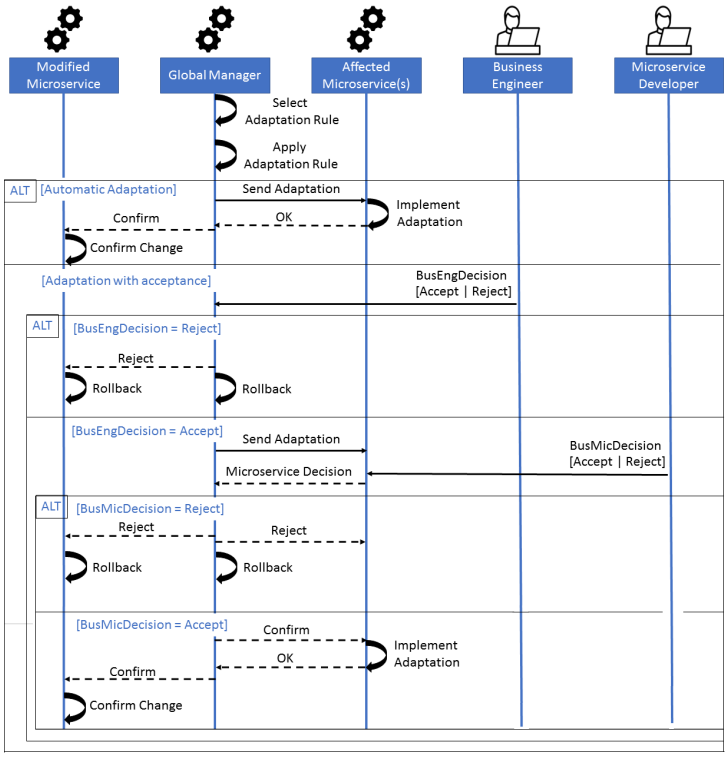

Figure 4: Phases of the evolution protocol

On the other hand, an adaptation rule is classified as *adaptation with acceptance* when the adaptation required to maintain the integrity of the choreography can be automatically applied in the BPMN collaboration diagram that represents the *global composition view*, but the adaptation implies some alterations in the *coordination requirements* among microservices. For instance, the adaptation may imply changing the execution order of some microservices (see the example explained in Figure 2). In this case, *functional requirements* are kept (i.e., all the tasks remain after the change), but the flow of these tasks changes. Thus, human participation is needed. Therefore, when the adaptation rule has been selected and applied on the BPMN collaboration diagram that represents the *global composition view*, stored in the *Global Manager*, the BP engineer must decide if the adaptation rule is accepted or not. If the adaptation rule is rejected, the change is denied and the modified microservice and the *Global Manager* must perform a rollback to return to their prior state before the introduction of the change. If the adaptation is accepted the BPMN diagram is re-split into BPMN fragments, and they are sent to the corresponding microservices. In order to implement the adaptation rule in each affected BPMN fragment, the adaptation must be accepted by each microservice developer affected by the change. If one of them rejects the adaptation, the change is denied and the modified microservice and the *Global Manager* must perform a rollback to return to their previous state before the application of the change. In addition, the *Global Manager* informs the rest of the affected microservices that the adaptation has been rejected, and thus they cannot implement it. If everyone accepts the adaptation, a commit message is sent to the modified microservice and the affected ones to inform them that the adaptation has been accepted and consequently, the introduced change and the adaptation rule can be implemented.

6. Related work

In the area of service compositions, [10] provides a formal method for representing syntactic properties of orchestrations and a set of axioms/invariants to align the orchestration model in a syntactic and

semantic way. In their work, the orchestration and the choreography are considered two different elements with different representations and consequently, to propagate the changes it is required to apply transformations between the two models. Our proposal implements an architecture to automate the evolution process when a change is introduced from the perspective of one microservice, and we also make it easier to understand the impact that the change and the adaptations have on the composition, since we have a visual representation of the global composition and the BPMN fragments.

In the area of flexible business process [11] proposes an approach to model business scenarios as a set of small fragments and the use of data objects states to combine them at runtime. Their work differs from ours in that they cover major changes in fragments such as adding new one, while we consider changes that affect specific elements inside the process of a BPMN fragment, such as tasks or interaction elements (Throw/Catching Events).

Finally, in the area of change management, [12] proposes change propagation algorithms to ensure behavioral and structural soundness of choreography partners in their processes. We go a step further and propagate and adapt the changes to guarantee the participation of every partner in the composition. [13] explains a negotiation phase to apply a change, but no mechanism is proposed to ensure that all partners have applied the change. [14] presents an approach to apply incremental changes (modify, add, and delete) to the participants of the choreography, but it does not consider mechanisms to ensure the propagation of the changes. Our approach follows a protocol that ensures that local changes and adaptations are propagated and implemented in the global composition and to each affected participant. [15] defines a set of rules to ensure the correctness of change operations. Their work is a formalization of event-based process and its refinement, but it is not clear how this refinement can be done from a participant's perspective. Furthermore, their proposal has not been put into practice, while our proposal has been implemented.

7. Conclusions

This thesis work faces the challenge of supporting the bottom-up evolution of a microservice composition, focusing on the *coordination requirements* found in distributed compositions. To do so, we introduce a characterization of the changes that can be performed from the local perspective of a microservice and the creation of a catalogue of adaptation rules that can be applied to the rest of microservices in order to maintain the functional integrity of the composition. In addition, it proposes an extension to an existing microservice architecture to integrate a MAPE-K loop for the selection of adaptation rules when local changes are introduced. This new component is supported with a protocol to allow the acceptance of developers, when necessary, when an adaptation rule is selected by the MAPE-K component, and it also guarantees the propagation of the changes and the adaptations to the rest of the participants of the composition.

To finish this thesis work, we are extending the characterization of the changes and the catalogue of adaptation rules by considering new adaptation rules to support local changes that affect the data exchanged by the microservice. Consequently, the MAPE-K component must be re-trained to integrate new adaptation rules. In addition, we want to study the application of online machine learning techniques so the classifier used in the planning phase of the MAPE-K loop can be automatically re-trained from the different results obtained progressively. Finally, we also plan to validate the protocol and the microservice architecture with more complex experiments to verify the effectiveness of the tools developed in environments with multiple microservices.

Acknowledgements

This work is part of the R&D&I project PID2020-114480RB-I00 funded by MCIN/AEI/10.13039/501100011033. It is also supported by the Research and Development Aid Program (PAID-01-21) of the UPV.

References

[1] M. Weske, Business Process Management: Concepts, Languages, Architectures. Springer. (2007).

[2] J. Lewis and M. Fowler: Microservices, 2014. URL: https://martinfowler.com/articles/micro-services.html (accessed April 2023).

[3] Weber, B., Reichert, M., & Rinderle-Ma, S., Change patterns and change support features–enhancing flexibility in process-aware information systems, Data & knowledge engineering 66.3 (2008) 438-466.

[4] I. Beerepoot, C. Di Ciccio, H. A. Reijers, S. Rinderle-Ma, W. Bandara, A. Burattin and F. Zerbato, The biggest business process management problems to solve before we die. Computers in Industry 146 (2023) 103837.

[5] P. Valderas, V. Torres, and V. Pelechano, A microservice composition approach based on the choreography of BPMN fragments, Inf. and Soft. Technology 127 (2020) 106370.

[6] J. Ortiz, V. Torres, and P. Valderas, Formalisation of Evolution issues in Microservice Compositions implemented as a choreography of BPMN Fragments, Research Report. URL: https://microservicere-search.github.io/AdaptationRuleCatalogue.pdf (accessed May 2023).

[7] J. Ortiz, V. Torres, and P. Valderas, Microservice compositions based on the choreography of BPMN fragments: facing evolution issues, Computing 105.2 (2022) 1-42.

[8] J. Miao, and L. Niu, A Survey on Feature Selection, In Procedia Computer Science 91 (2016) 919–926.

[9] D. Butler, D. Aspinall, A. Gascón, On the formalisation of Σ-protocols and commitment schemes, In Principles of Security and Trust (2019) 175–196.

[10] A. Wombacher, Alignment of choreography changes in BPEL processes, In IEEE International Conference on Services Computing (2009) 1-8.

[11] M. Hewelt, and M. Weske, A hybrid approach for flexible case modelling and execution, in: Proceedings of the 14th. Business Process Management Forum: BPM Forum, Rio de Janeiro, Brazil, 2016, pp. 38-54.

[12] W. Fdhila, C. Indion, S. Rinderle, and M. Reichert, Dealing with change in process choreographies: Design and implementation of propagation algorithms, Information systems 49 (2015) 1-24.

[13] W. Fdhila, C. Indiono, S. Rinderle-Ma and R. Vetschera, Multi-criteria decision analysis for change negotiation in process collaborations, In IEEE 21st International Enterprise Distributed Object Computing Conference (2017) 175-183.

[14] R. R. Mukkamala, T. Hildebrandt, and T. Slaats, Towards trustworthy adaptive case management with dynamic condition response graphs, In 17th IEEE International Enterprise Distributed Object Computing Conference (2013) 127-136.

[15] S. Debois, T. Hildebrandt, and T. Slaats, Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes, in: Proceedings of the 20th. Formal Methods: 20th International Symposium, Norway, 2015, pp. 143-160.

[16] F. Fakhfakh, H. H. Kacem, & A. H. Kacem, Ensuring the correctness of adaptive business processes: a systematic literature review, International Journal of Computer Applications in Technology 62.3 (2020) 189-199.

[17] A. J. V. Vaca, & R. M. Gasca. Opbus, Fault tolerance against integrity attacks in business processes, In: Proceedings of the 3rd. Computational Intelligence in Security for Information Systems, Berlin, Germany, 2010, pp. 213-222.

[18] C. Larman, V.R. Basili, Iterative and incremental development: A brief history, Computer 36.6 (2003), 47–56.

[19] S. Jahan, I. Riley, C. Walter, R. F. Gamble, M. Pasco, P. K. McKinley, & B. H. Cheng, MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases, Future Generation Computer Systems 109 (2020) 197-209.

[20] D. Garlan, S. W. Cheng, & B. Schmerl, Increasing system dependability through architecture-based self-repair, Architecting dependable systems (2003), 61-89.

[21] K. Czarnecki, S. Helsen, Classification of model transformation approaches, In: Proceedings of the 2nd. OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, USA, 45.3, 2003, pp. 1–17.