Computing (2023) 105:375–416
https://doi.org/10.1007/s00607-022-01128-8

REGULAR PAPER

Microservice compositions based on the choreography of BPMN fragments: facing evolution issues

Jesus Ortiz[1] · **Victoria Torres**[1] · **Pedro Valderas**[1]

Received: 28 April 2022 / Accepted: 18 October 2022 / Published online: 12 November 2022

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 characteristic. In this case, the coordination of such fragmented BPs is usually achieved by means of event-based choreographies. 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. We face the challenge of evolving a microservice composition that is globally defined in a BPMN model but executed through a choreography of BPMN fragments. We introduce a protocol to manage the propagation of a change done by one microservice to be integrated into both the BPMN fragments of the rest of the microservices and the global BPMN model. This protocol also supports the negotiation among participants and the automatic suggestion of model adaptations to maintain the functional integrity of the composition. These suggestions are supported by a catalogue of adaptation rules that precisely characterize every possible change and propose actions to be considered by the affected microservices. All the evolution process is done at the modelling level, without managing hard-coded implementations. We have developed specific tools to facilitate the practical adoption of this protocol, and we have validated our work in an experiment with users. We can conclude that the proposed approach is effective to evolve microservice

✉ Pedro Valderas
pvalderas@pros.upv.es

Jesus Ortiz
jortiz@pros.upv.es

Victoria Torres
vtorres@pros.upv.es

[1] PROS Research Center, Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain

compositions implemented as event-based choreography of BPMN fragments from the local perspective of one partner.

Keywords Microservices · Composition · Evolution · Protocol · Bottom-up

Mathematics Subject Classification 68U35

1 Introduction

Business Processes (BPs) are the key instrument to organizing and understanding the interrelationships of the different activities required to produce an outcome for the market [1]. Such BPs are specified as models which can be described following either an orchestration or a choreography approach. While orchestrations are governed by a centralized control flow, choreographies are governed by the interaction that occurs among the involved parties through the exchange of messages [2]. When BPs activities are performed in a decentralized way, e.g., by different departments within the same organization, the choreography approach turns more convenient since it puts the focus on the collaboration that occurs between the involved partners through the exchange of messages. Besides, the decoupling characteristic of microservices makes them a very interesting and convenient way to implement such processes. Microservice architectures [3] 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 [4].

However, choreographies split the control flow of compositions among the different participant microservices, which makes them hard to analyse and understand when requirements change. Our previous work [5] faces this problem and proposes an approach based on the choreography of process fragments defined by the Business Process Model and Notation (BPMN)[6] to address it. According to this approach, BP engineers create the big picture of the microservice composition through a BPMN model. Then, this model is split into BPMN fragments which are executed through an event-based choreography. This composition approach is supported by a microservice architecture developed to achieve that both descriptions of a microservice composition, the big picture and the split one, coexist in the same system. This solution introduces two main benefits regarding the microservice composition. On the one hand, it facilitates BP engineers to analyse the control flow if the composition's requirements need to be modified. On the other hand, it provides a high level of decoupling in the execution of microservices, allowing the independent management of the BPMN fragments by the corresponding development team.

However, this solution introduces a new challenge to be faced: how to evolve a microservice composition that is globally defined following an orchestration schema in a BPMN model, but which is executed through the choreography of several process fragments. Changes in process-based systems have been identified as crucial in most

application domains [7–10]. Changes can be needed due to several reasons such as the advent of new regulations or the emergence of new competitors in the market that forces the supported requirements to be adapted. In this work, we face the evolution of microservice compositions that are described in a global model, as it is done when an orchestration approach is followed, but which is split into model fragments that are distributed through microservices and executed through an event-based choreography. In addition, this evolution is faced at the modelling level, allowing changes to be managed through descriptions of a high level of abstraction such as BPMN models, instead of having to manage hard-coded implementations of the control flow and the interchange of messages.

1.1 Previous work: composition of microservices

To properly understand our current work, this section introduces a summary of our previous work [5] by applying the proposed microservice composition approach to a representative example, which is used as a motivating example in the rest of the paper. This approach proposed two main steps to create a microservice composition: (1) to create the big picture of the composition in a BPMN model and (2) to split it into BPMN fragments that will be deployed into the corresponding microservices and executed through an event-based choreography. We consider a scenario based on the e-commerce domain, which describes the process for placing an order in an online shop. To support this process, we need to consider different business responsibilities related to the management of customer information, the control of the inventory of products, the processing of the payment, and the shipment of products. We propose to create a microservice to support each of these business responsibilities: *Customers*, *Inventory*, *Payment* and *Shipment*. Figure 1 shows the big picture of this process represented in BPMN, which is created in the first step of the proposed approach. Note that each microservice is defined by a BPMN pool.

The sequence of steps that the microservices must perform is the following (see Fig. 1):

1. The *Customers* microservice checks the customer data and logs the request. If the customer data is not valid, the process of the order is cancelled. On the contrary, this microservice transfers the control flow to the *Inventory* microservice.
2. The *Inventory* microservice checks the availability of the ordered items. If there is not enough stock to satisfy the order, the process of the order is cancelled. On the contrary, this microservice books the requested items and transfers the control flow to the *Payment* microservice.
3. The *Payment* microservice processes the payment with the customer. If the payment fails, the process of the order is cancelled, and the control flow is transferred to the *Inventory* microservice. On the contrary, the control flow is directly transferred to the same microservice, without cancelling the purchase order.
4. If the payment has not been correctly processed the *Inventory* microservice releases the products and the process finishes. If the payment is OK, the *Inventory* microservice updates the stock of the purchased items and the control flow is transferred to the *Shipment* microservice.

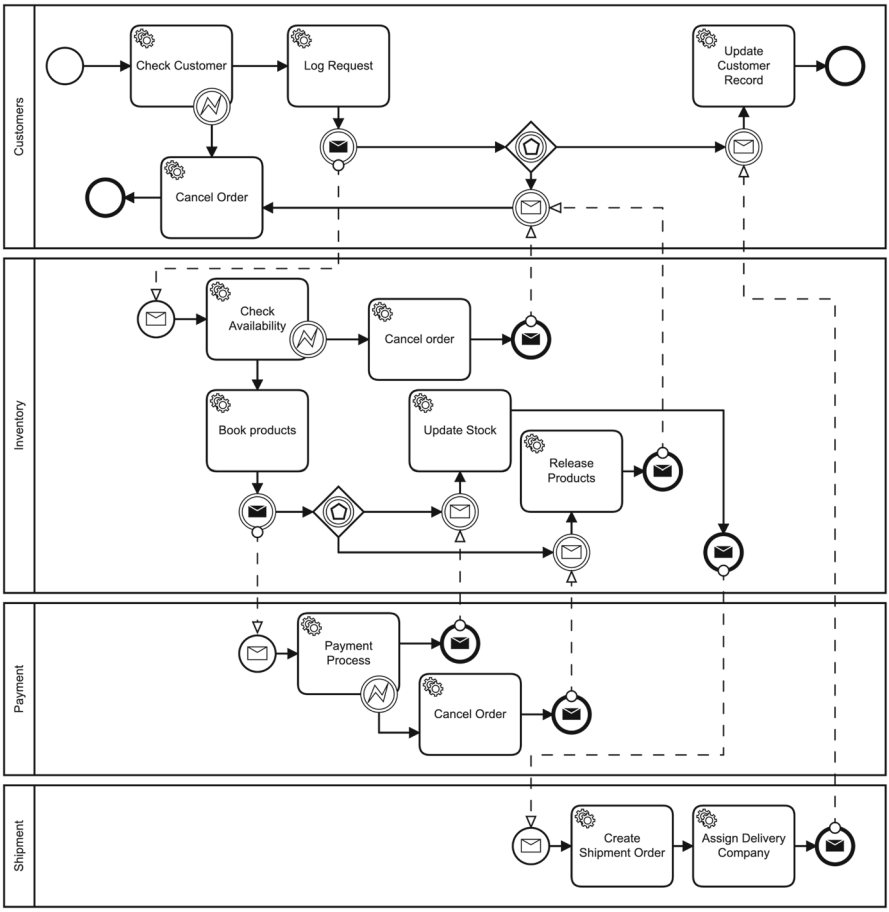

Fig. 1 Big picture of a microservice composition

5. The *Shipment* microservice creates a shipment order and assigns it to a delivery company. Then the control flow is transferred to the *Customer* microservice.
6. Finally, the *Customer* microservice updates the customer record and informs the customer about the shipment details. Afterwards, the process finishes.

After creating the big picture of the composition, the second step consists in splitting it into BPMN fragments that describe the functional responsibility of each microservice. This is done automatically by a tool we developed [5]. At runtime, each microservice oversees executing its corresponding BPMN fragment and informing the other participants about it through publishing asynchronous events in a communication bus. In this way, the microservice composition was executed by means of an event-based choreography of BPMN fragments in which microservices wait for an event to execute its corresponding piece of work. This is shown in Fig. 2. Note how a microservice does not transfer the control flow to another microservice explicitly. Instead, a microservice publishes an event in a bus (depicted by solid blue arrows) to

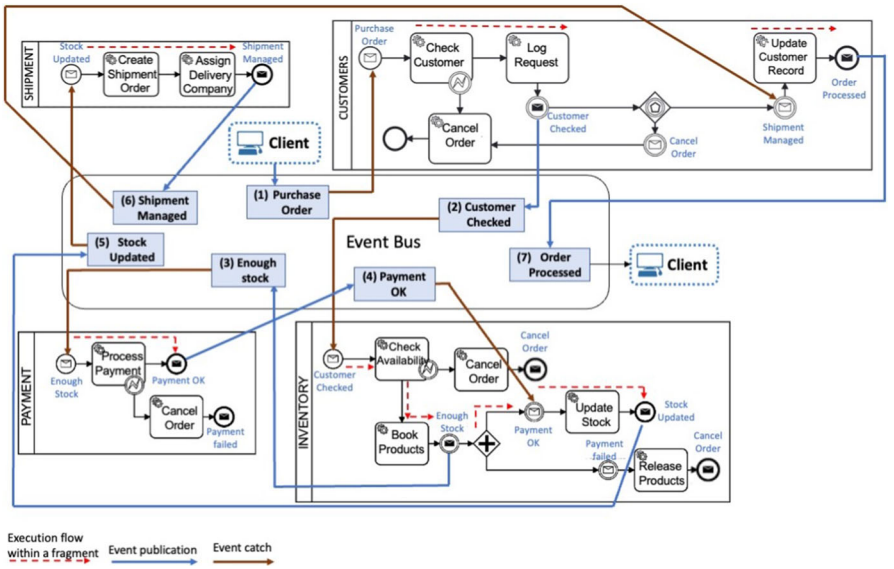

Fig. 2 Event-based choreography of BPMN fragments

indicate that a piece of work is completed, and the microservice that is waiting for this event (depicted by solid brown arrows) starts the execution of its BPMN fragment.

This composition approach is supported by a microservice architecture developed to achieve that both descriptions of a microservice composition, the big picture and the split one, coexist in the same system. In addition to the business microservices that participate in the composition (i.e. *Customers*, *Inventory*, *Payment*, and *Shipment*), this architecture introduces the *Global Manager* microservice (see Fig. 3) whose goal is to support the management of the BPMN model with the big picture of the composition, as well as splitting it into fragments and distributing the fragments among the microservices that participate in the composition.

1.2 Motivation: evolution of a microservice composition

The architecture proposed in our previous work introduces two approaches to evolve a microservice composition (see red arrows in Fig. 3): a top-down approach and a bottom-up approach.

By following a top-down approach the microservice composition is evolved by BP engineers from the BPMN model that represent the big picture. The evolution is done from a global perspective and the modifications introduced in the big picture are propagated to the corresponding BPMN fragments of each microservice. The propagation process is the same as when a composition is created, split, and distributed. Thus, this evolution strategy is natively supported by our previous work.

By following a bottom-up approach the microservice composition evolves from the BPMN fragments of individual microservices. In this case, the evolution is done from the local perspective of a specific microservice. This means that developers of

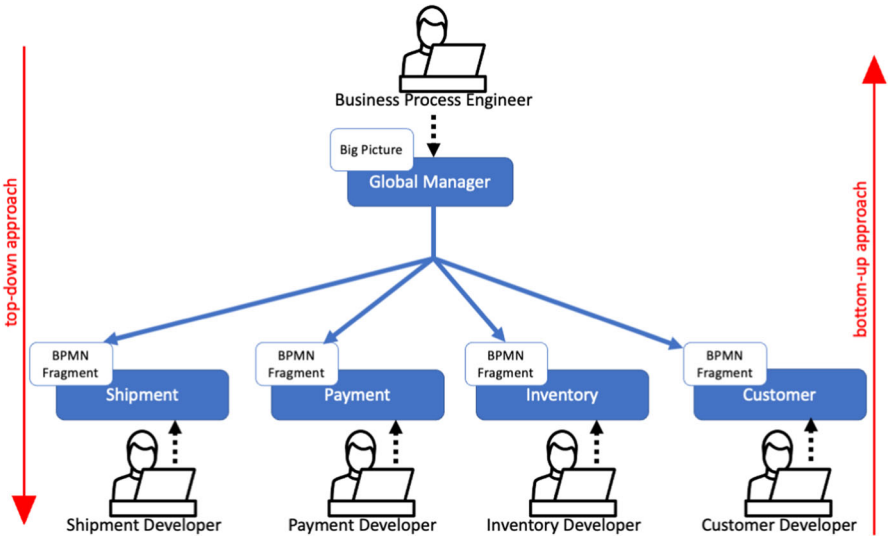

Fig. 3 Microservice architecture in which coexists the big picture of a composition and the split version

a microservice can modify the BPMN fragment under its responsibility as long as they integrate the changes with the rest of the system, i.e., the BPMN fragments of the rest of the microservices and the BPMN model with the big picture managed by the *Global Manager* microservice. Note that allowing local changes in a microservice composition reinforces the independence among development teams that is demanded by this type of architecture, but at the same time may compromise the integrity of the whole composition. For instance, let us consider that the developers of the *Customers* microservice decide to modify its BPMN fragment (see Fig. 2) in such a way that the event "Customer Checked" is not published anymore. Then, the *Inventory* microservice, which is waiting for it, will never start and execute its tasks, and then, the microservice composition will never continue. Thus, a local modification in a BPMN fragment stops prematurely the global composition, making it impossible to achieve the objective for which it was initially designed.

Considering that microservice compositions are defined through a BPMN business process, this problem can be contextualised within the area of flexibility in Business Process Management (BPM) [1], which has become a centre of attention from both commercial and research institutions as an understanding of requirements for BPs [11]. Flexibility also relates to the evolution of BPM, especially because of its ability to adapt BPs to predicted and unpredicted real-world changing scenarios. Consider, for instance, unpredictable changes due to altered legal requirements.

Many solutions have been proposed in the literature to support BP flexibility [12]. All of them can be classified into one of the following four categories proposed by [13]: (1) Variability, which is the ability to derive different variants from the same BP; (2) Adaptation, which is the ability to temporarily deviate the flow during the execution of a BP; (3) Looseness, which is the ability to execute a BP with some decisions that affect the control flow and are not fully defined or undefined; or (4)

Evolution, which is the ability to permanently modify a BP affecting all future BP enactments. Our solution is classified into the last category.

The survey conducted in [12] shows that many of the works that face BP flexibility focus on variability, adaptation, or looseness. Little attention is paid to the evolution of BP (only 4 works out of 70 explicitly mention the evolution as a reason to lead the definition of a solution for flexibility management). Thus, the consideration of BP evolution is identified as a research direction to be considered

In addition, BP evolution is usually faced from an orchestration perspective, in which the BP is defined in a centralized description that needs to be globally evolved. As we discuss further when analysing the related work (Sect. 6), few solutions face the evolution of choreographed BPs from the specific perspective of one participant, as we do in this work. This type of evolution has the additional complexity introduced by the interaction of autonomous and independent partners. In the context of microservices, despite many solutions support their composition based on BPMN and other well-known BP modelling languages, their evolution is mostly not faced.

1.3 Problem statement

Considering the motivation presented above, this paper presents a solution to face the evolution of a microservice composition from the local perspective of a microservice, which can be stated by the following research question:

How can we evolve a microservice composition based on the choreography of BPMN fragments from the local perspective of a microservice without compromising the integrity of the whole composition?

1.4 Main objectives and contributions

The main objective of this work is to answer the above-introduced research question by providing a solution that supports the evolution of a microservice composition from the local perspective of one participant. This solution should facilitate the propagation of local changes through the *Global Manager* and the rest of the microservices; should automate, as much as possible, the suggestion of adaptations to the microservices affected by the local change; and should support the manual negotiation of an adaptation when developers do not consider the proposed adaptation adequate. To achieve this, the contributions of this paper are the following:

1. A protocol that allows us to propagate and communicate local modifications within a microservice composition in order to automate, when possible, the compensation actions required to maintain the composition integrity.
2. A characterization of the changes that can occur from the local view of a microservice, analysing the impact that each change has on the global composition, and proposing compensation actions that ensure, when possible, the achievement of the global composition goal.
3. A BPMN-based tool that supports microservices developers to perform local modifications according to the proposed protocol.

In previous works, we presented an initial attempt to categorize local changes at the microservice level [14] and introduced a preliminary version of the protocol [15]. Based on these first efforts, in this paper, we take a step forward and present a more precise characterization of local changes. While the previous works only consider delete actions, this paper considers also update and create actions. In addition, it also introduces a redefined version of the protocol in which we clearly specify both human and software participants, redefine the Global Manager microservice's involvement to have greater responsibility for analysing local changes, and add a new phase to allow the negotiation of the suggested adaptations. In addition, we present the tool we have developed to support the bottom-up approach and the evaluation we have carried out through a practical experiment to assess both, the protocol and the supporting tool.

1.5 Research methodology and paper structure

A methodology in line with the precepts of Design Science Research (DSR) [16, 17] is used for this research project. DSR aims at developing practical solutions that can be used by professionals in their field. More concretely, solutions - or design artefacts - can take the form of constructs, models, methods, or instantiations [16]. Considering the contributions presented in Sect. 1.4, the artefact we have developed is a protocol to support a bottom-up evolution of a microservice composition that facilitates the propagation of changes, the suggestion of adaptations, and their negotiation. According to [16], this artefact can be classified as a method since it defines a set of actionable instructions that are conceptual and not algorithmic.

We applied the DSR methodology to develop this artefact and performed the six activities proposed in [17] by following a problem-centred approach. These six activities are: (1) Problem identification and motivation; (2) Define the objectives for a solution; (3) Design and development; (4) Demonstration; (5) Evaluation; and (6) Communication. Thus, we first identified the specific research problem and motivated it, which was presented in Sects. 1.1, 1.2 and 1.3. This motivation is complemented by the study of the state of the art that is presented in Sect. 6, which compares the improvements introduced by our solution with pre-existing ones. Next, we defined the objectives of the solution, which have been presented in Sect. 1.4. The next activity in the DSR methodology consists of the design and development of the artefact required to support the proposed objectives. This is explained in Sects. 2 and 3, which introduce the protocol proposed to evolve a microservice composition when a participant introduces a local change, and a catalogue of adaptation rules to be used within the protocol to automatically suggest adaptations to the affected microservices. To do so, we followed an action-research development [18] in such a way we iteratively study the problem to solve, apply some actions, and analyse if the obtained results satisfy our purposes.

The fourth activity to be performed is the demonstration, in which the developed artefact must be used to solve one or more instances of the considered problem. To do so, we extended the architecture that supports the composition of microservices with the required tool support and developed a proof-of-concept prototype to test the feasibility of the proposed protocol with the running example. This is explained in

Sect. 4. The implementation of the required tools was done by following an iterative and incremental process [19] in such a way the tools were progressively developed and tested with examples, refining previous implementations when some errors were detected.

The next activity proposed by the DSR methodology is the evaluation, in which we must observe and measure how well the artefact supports a solution to the problem. To do so, we arranged a controlled subject-based experiment [16] in order to evaluate the effectiveness of the developed artefact to allow users to solve an instance of the problem considered in this work. The experiment was conducted by applying the guidelines proposed in [20]. It is presented in Sect. 5. To complete the DSR methodology we are communicating our results to the research community through this paper, whose last section presents some conclusions and provides insights into directions for future work.

2 The proposed evolution protocol

In general, when a process of a system is changed, it must be ensured that structural and behavioural soundness is not violated after the change [9]. When the process is supported by a choreography, additional aspects must be guaranteed due to the complexity introduced by the interaction of autonomous and independent partners. As exemplified above, when a participant introduces some change in its part of the process, it must be determined whether this change affects other partners in the choreography as well. If so, the change must be propagated to the rest of the partners which may involve performing adaptations to maintain the consistency and compatibility of the choreography. The decision of whether to adopt these adaptations must be left to partners and may be subject to negotiations, which can be costly and time-consuming.

This section proposes an evolution protocol to facilitate the propagation of changes, the suggestion of adaptations, and their negotiation in the context of our microservice composition approach that is based on the choreography of BPMN fragments. The main goal of this protocol is to achieve maximum automation in these activities in order to synchronize the changes done in the BPMN fragment of a microservice with both the BPMN fragments of the rest of the microservices and the big picture managed by the *Global Composition* microservice. To do so, this protocol uses the adaptation rules presented in Sect. 3 which are implemented in the supporting web tools introduced in Sect. 4.

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 [21]. The participants involved in the protocol that we propose can be either software participants, represented by microservices which perform actions automatically (i.e., the locally modified microservice, the rest of the microservices that participate in the process composition, and the *Global Manager* microservice), 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 above microservices, and the BP engineers responsible for the *Global Manager* microservice). Regarding the actions and the interchange of messages considered by

the protocol, they are grouped into four main phases: (1) classification of the local change; (2) propagation of a coordination change; (3) suggestion and realization of an adaptation; and (4) negotiation.

Note that the impact of change propagation on running instances is not addressed by the proposed protocol. The changes generated by the protocol application only affect the new instances created after. This decision is technologically supported by the default behaviour of the BPMN engine that is used to execute microservices compositions (Camunda, further details are given in Sect. 4.1), which provides a versioning strategy to evolve process definitions without affecting running instances. However, we plan to consider the evolution of running instances in further works. This will imply both adapting the technological solution implemented to support the evolution of microservice compositions and revising the proposed protocol in order to analyse the implications of this issue and adapt (if needed) the proposed steps accordingly.

Next, we introduce the four phases of the protocol in detail.

2.1 Phase 1: classification of the local change

A BPMN fragment describes two types of requirements:

- **Functional requirements**, which are represented by the BPMN tasks that are defined in the fragment of each microservice. They define the actions that each microservice does in the context of a composition but independently from the rest of the microservices. These represent the internal business logic of the microservice. Changes in these requirements imply isolated changes in the functional responsibilities of the microservice. This type of local change can be applied without any synchronization action with the rest of the microservices since they do not have any impact on them. Therefore, they just need to be integrated into the big picture of the composition.
- **Coordination requirements**, which define how microservices communicate with each other to achieve the goal of the composition they participate in. Note that, at runtime, this communication is done by means of an asynchronous event bus (see Fig. 2). Thus, these requirements are represented by the BPMN elements that define an event-based communication, i.e., by throw and catching events as well as the flows used to connect them. The modification of coordination requirements may produce inconsistencies in the composition since some events required by other microservices to start may not be produced.

The first phase in the proposed protocol includes several steps to classify the local change done by a microservice developer into any of these two types of requirements (see Fig. 4).

If the local change only affects functional requirements, it can be confirmed by the modified microservice without any synchronization action with the rest of the microservices. The locally modified microservice just needs to send the functional changes to the *Global Manager* microservice to be integrated into the big picture. Then, the application of the protocol finishes. If the local change affects coordination requirements, they are also sent to the *Global Manager*, but additional efforts may be

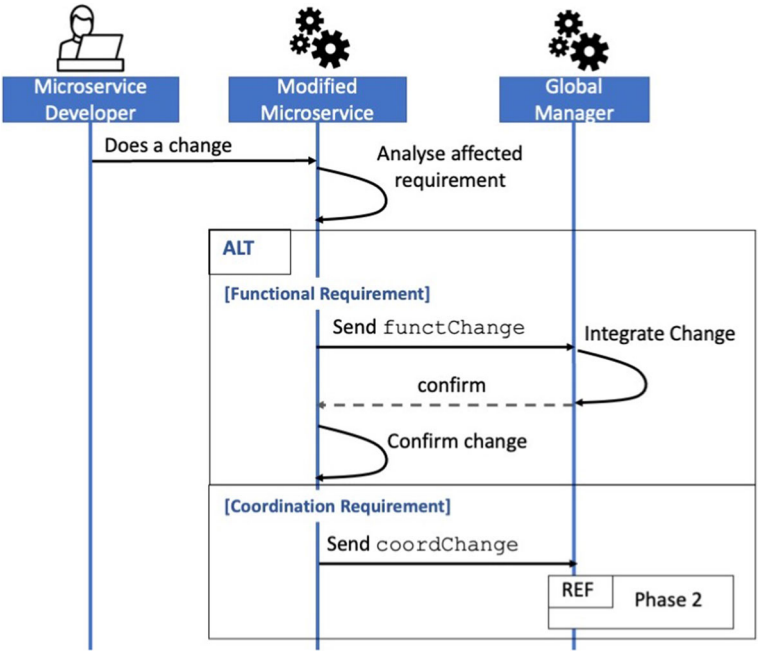

Fig. 4 Phase 1 of the evolution protocol

required to synchronize it with the rest of the participants, which are managed in the following phases of the proposed protocol.

2.2 Phase 2: propagation of a coordination change

If a local modification is classified as a change that affects coordination requirements (i.e., the communication defined between BPMN fragments), the *Global Manager* must initially identify the type of local modification performed (see Fig. 5). This modification can be of three types: (1) add, (2) delete, and (3) update an event-based communication element in a BPMN fragment. Note that we consider that a local modification of an event-based communication element can affect either the name of the event that is being sent/received or the data that optionally can be attached to this event (which we internally represent as a set of tuples key-value). In this sense, we consider the actions of adding and deleting an event-based communication element with and without data; and the action of updating the name of the event, the attached data, or both. Depending on the action and whether the event has attached data a different adaptation is proposed (which is based on the catalogue of adaptation rules introduced in Sect. 3).

Regarding the inconsistencies produced by the modification of an event-based communication element, note that adding a new BPMN throwing event does not produce any inconsistency in the global composition since coordination requirements are not changed but extended. In fact, this type of change introduces new coordination pos-

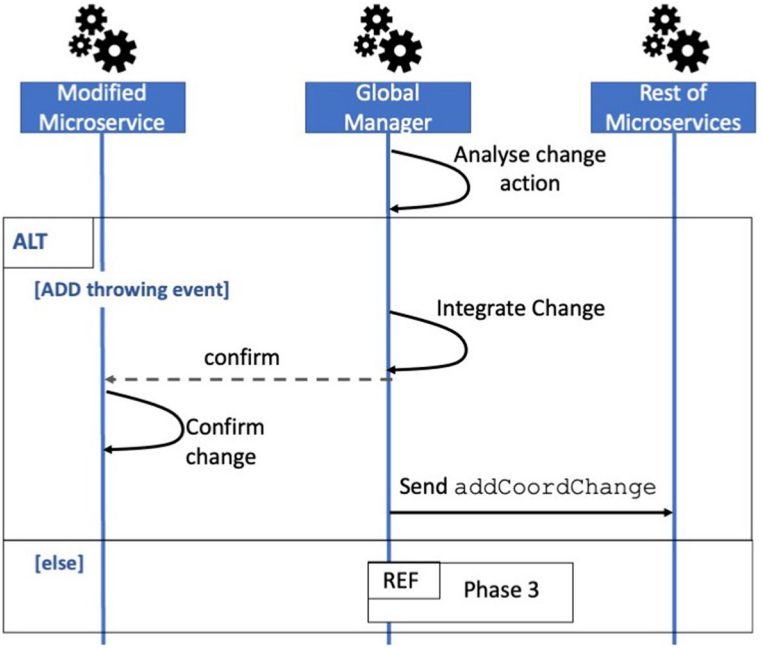

Fig. 5 Phase 2 of the evolution protocol

sibilities among microservices. Thus, (1) the *Global Manager* integrate the changes in the big picture; then, (2) the modified microservice can confirm the change in its BPMN fragment; and finally (3) the *Global Manager* sends the change to the rest of the participants in order to inform them about the new coordination possibilities. In this case, the protocol finishes at this point.

On the contrary, deleting or updating an event-based communication element, or creating a new catching event-based communication element may produce inconsistencies in the composition since some events required by other microservices may not be produced. In these situations, inspired by the two-phase commit protocol used in distributed database transactions [22], the microservice that performs the local change in its BPMN fragment cannot confirm it until the affected participants and the *Global Manager* reacts (either positively or negatively) to it. This is managed in Phase 3.

2.3 Phase 3: suggestion and realization of an adaptation

When a delete or update coordination change is sent to the *Global Manager*, this microservice automatically analyses the change and generates an adaptation for those other microservices that are affected by the change. To do so, the *Global Manager* microservice uses a catalogue of rules based on the characterization of changes (see Sect. 3). This change characterization (1) identifies the inconsistencies that a local change in a microservice produces in the rest of the participants, and (2) defines a process adaptation through the set of compensation actions that are required to maintain

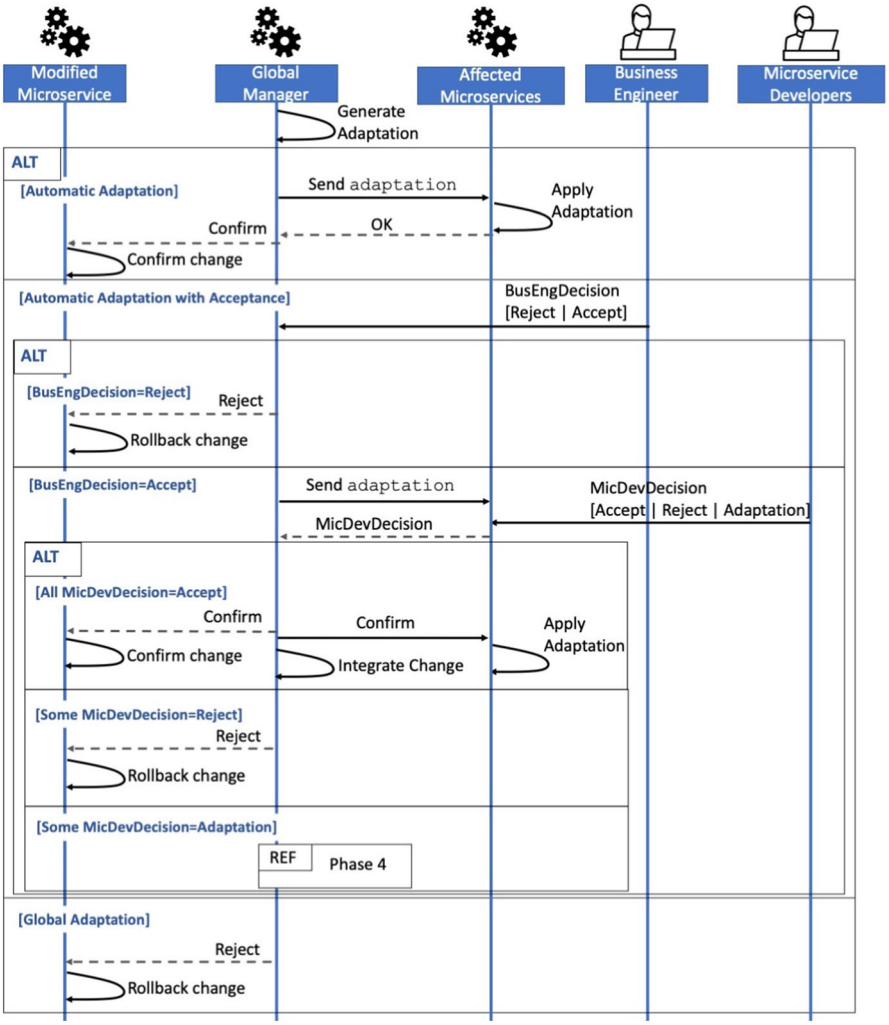

Fig. 6 Phase 3 of the evolution protocol

the integrity of the choreography. In addition, this characterization also classifies each adaptation as *Automatic adaptation*, *Automatic adaptation with acceptance*, or *Global adaptation*. Depending on this classification, the protocol proposes different actions (see Fig. 6).

An adaptation is classified as *Automatic adaptation* when compensation actions to maintain the integrity of the choreography can be automatically applied in the BPMN fragment of an affected microservice since functional and coordination requirements are both maintained. No human participation is required. For instance, if a microservice developer just updates the name of a published event (e.g., the event "Payment Ok" is replaced by "Available Credit"), the microservices that were waiting for this event can automatically adapt its BPMN fragment to update it to the new name. Thus,

affected microservices can automatically adapt their corresponding BPMN fragment and accept the change. Then the locally modified microservice can confirm the change and the protocol finishes.

An adaptation is classified as *Automatic adaptation with acceptance* when compensation actions can be automatically applied in the BPMN fragment of an affected microservice in order to support the functional requirements. However, the coordination among some microservices may change. For instance, a compensation action may imply changing the execution order of two microservices from sequential to parallel. In this case, functional requirements are kept (i.e., all the tasks remain after the change), but the flow of these tasks changes and some data may be missed for some microservices. Thus, human participation is needed. A manual acceptance by the business engineer and the developers of the affected microservices is required to confirm the proposed evolution. If all of them accept the suggested adaptation then the modified microservice can confirm the change and integrate it into the big picture. If some of them reject the suggested adaptation, the change done by the modified microservice must be rollbacked. Note that business engineers can manually modify the suggested adaptation before accepting it. A microservice developer can also propose an alternative adaptation for the BPMN fragment of its microservice. In this case, the protocol continues with Phase 4.

Finally, an adaptation is classified as *Global adaptation* when compensation actions to maintain the integrity of the choreography imply important modifications in both coordination and functional requirements. In this case, further analysis of the whole composition must be done by business engineers from a global perspective of the composition. In this case, the change is rejected and the microservice composition needs to be redefined from scratch [5].

2.4 Phase 4: negotiation

The last phase of the proposed evolution protocol is applied when the *Global Manager* suggests an adaptation classified as *Automatic adaptation with acceptance* and the microservice developer of an affected microservice does not accept it but proposes a new one. This is graphically represented in Fig. 7.

The new proposal of adaptation is sent to the *Global Manager* and the business engineer can accept or reject it. S/he can also finish the negotiation activity. Each time the business engineer rejects a new adaptation proposal, the microservice developer can propose a new one, accept the one initially sent by the business engineer, or reject it. If the microservice developer does not propose a new adaptation and rejects the one proposed by the business engineer, the change proposed by the modified microservice is rejected. It is also rejected if the business engineer finishes the negotiation activity. If the business engineer accepts a proposal of the microservice developer, or the latter accepts the proposal of the business engineer, then the adaptation is applied by the affected microservice, and the change is integrated into the big picture and confirmed by the modified microservice.

Finally, note that the developer of an affected microservice could send a proposal of adaptation that has an impact on other microservices that do not participate in the

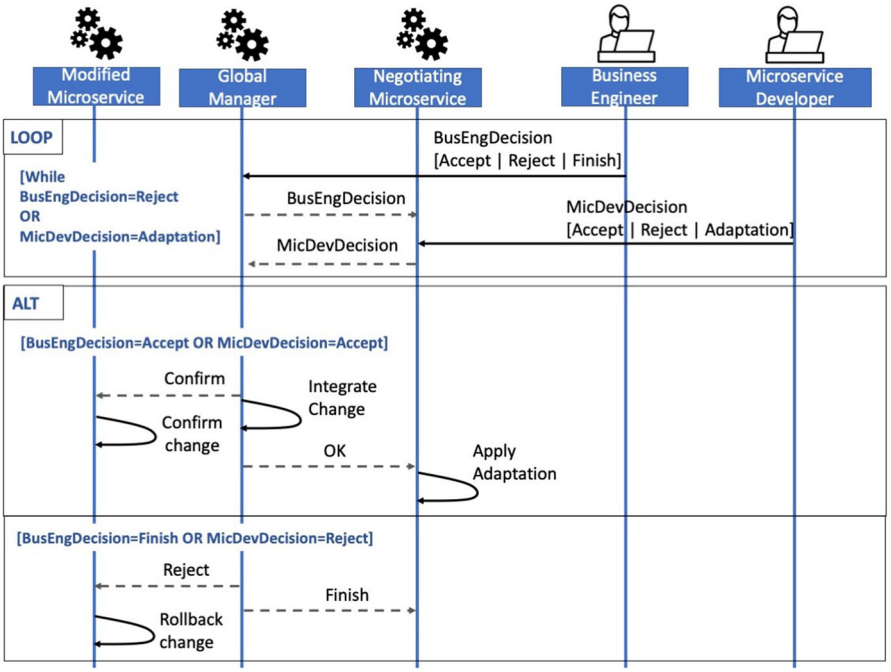

Fig. 7 Phase 4 of the evolution protocol

negotiation process. This is a complex scenario that needs multiple negotiations among the business engineer and the developers of several affected microservices (the one affected by the initial change and the others affected by the proposed adaptations). To face this challenge additional investigation is needed and it will be treated as further work. In the current version of the protocol, the business engineer should reject the proposed adaptation.

3 Adaptation rules to support local changes in coordination requirements

According to the protocol presented above, the *Global Manager* microservice automatically suggests an adaptation when it is informed about a local change that affects the coordination requirements of the process composition (in particular, when any event-based communication BPMN element is deleted or updated, or when a cathing event is added). This automatic behaviour has been implemented in the web tools that support the protocol (further presented in Sect. 4). To do so, we defined a catalogue of adaptation rules that exhaustively analyze the different scenarios in which an event-based communication BPMN element could be added, deleted or updated, and identified both, the impact of this change in the global choreography and the adaptation (in terms of compensation actions) that must be performed to maintain, when possible, the integrity of the choreography. Depending on the required compensation

actions, we also classified each adaptation rule into the types introduced above, i.e., *Automatic adaptation*, *Automatic adaptation with acceptance*, and *Global adaptation*.

As a result, a total amount of 19 rules have been defined. As representative examples here we just present some of the proposed adaptation rules which refer to the deletion (rules #1 to #4) and update (rule #9) of end and intermediate events under different conditions. However, the complete catalogue of rules has been defined in a research report accessible in [23]. Note that these rules differentiate between actions that modify an event-based communication BPMN element that either attaches some data or not. Those that do not attach data are used by microservices to notify the success or the failure of a piece of work. They allow defining the flow in which microservices must be choreographed to perform their actions. Those that attach data are also used to define the flow of the choreographed microservice composition, but they also carry data that some microservice generate to be processed by others.

3.1 Deleting a throwing event without attached data

This change implies the removal of a BPMN element that sends an event (a throwing event) to inform that a piece of work has been done, without data interchange. To support this change two adaptation rules are proposed. The affected microservices are those that have a catching event waiting for the message sent by the deleted element. Rule #1 adapts these microservices to start listening to the event triggered just before the deleted one during the choreography execution. However, note that one of the affected microservices could be also the one that triggers this event. Therefore, Rule #1 cannot be applied since a microservice should not listen to an event that is sent by itself. This affected microservice needs to be adapted in a different way and this is the reason why we need Rule #2, which is defined to be applied when the event just before the deleted one is triggered by a microservice affected by the delete action.

RULE #1

- **Affected element:** End Message Event or Intermediate Throwing Event.
- **Change:** Delete the affected element.
- **Conditions:**
  - The deleted event does not include data (Rule #3 or #4 are applied instead).
  - The event triggered before the deleted one is not generated by the affected microservice (Rule #2 is applied instead).
- **Affected microservice(s):** Those that have a catching event waiting for the message sent by the deleted element.
- **Generated inconsistency:** Affected microservices will never start or continue (creating a deadlock) since their execution depends on the triggering of the event that is just deleted.
- **Compensation actions:** Modify the affected microservices to wait for the event triggered before the deleted one.
- **Impact of the application:** Functional requirements are maintained. Coordination requirements are altered (the flow of some tasks changes from sequential to parallel).
- **Adaptation type:** Automatic adaptation with acceptance.

A representative example of the change supported by Rule #1 is removing the BPMN Intermediate Throwing Event "Stock Updated" in the BPMN Fragment of the *Inven-*

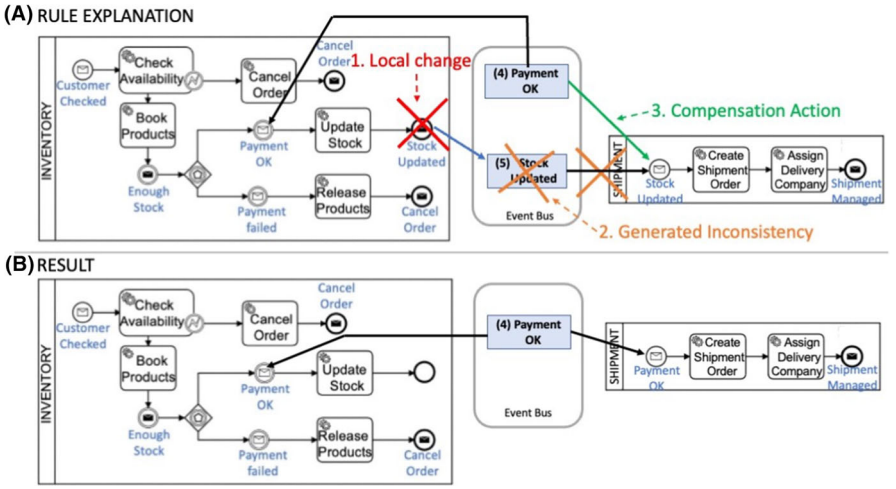

Fig. 8 Example of Adaptation Rule #1

tory microservice (see Fig. 8A).If this event is removed (see Fig. 8A), the *Shipment* microservice, which is waiting for it, will never start the execution of its BPMN fragment, and therefore, the microservice composition will never finish.

In order to allow the *Shipment* microservice to complete its tasks, it can be modified to wait for the event previously caught in the modified fragment, i.e., to wait for the "Payment OK" event. However, two microservices that initially performed some of their tasks in a sequential way (e.g., first *Inventory* updates the stock and then *Shipment* creates a shipment order) now are performed in parallel (e.g., after the local change, both the update of the stock and the creation of the shipment order are executed when the "Payment OK" event is triggered, see Fig. 8B). Thus, a manual confirmation by the business engineer and the Shipment developer is needed.

RULE #2

- **Affected element:** End Message Event or Intermediate Throwing Event.
- **Change:** Delete the affected element.
- **Conditions:**
  - The deleted event does not include data (Rule #3 or #4 are applied instead).
  - The event triggered before the deleted one is generated by the affected microservice (Rule #1 is applied otherwise).
- **Affected microservice(s):** Those that have a catching event waiting for the message sent by the deleted element.
- **Generated inconsistency:** Affected microservices will never start or continue (creating a deadlock) since their execution depends on the triggering of the event that is just deleted.
- **Compensation actions:** Delete the catching event that is waiting for the removed event in the affected microservices.
- **Impact of the application:** Functional requirements are maintained. Coordination requirements are altered (the flow of some tasks changes from sequential to parallel).
- **Adaptation type:** Automatic adaptation with acceptance.

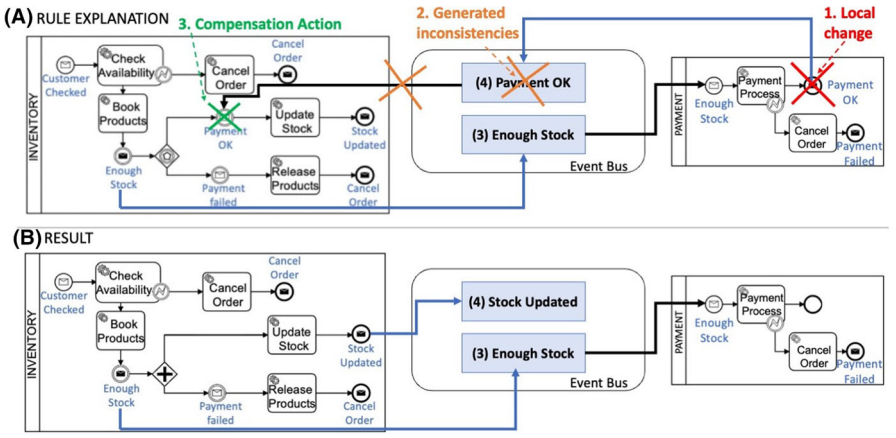

Fig. 9 Example of Adaptation Rule #2

A representative example of the change supported by Rule #2 is removing the BPMN Message End Throwing Event "Payment OK" of the *Payment* microservice (see Fig. 9A). In this case, the *Inventory* microservice, which is waiting for it, will never continue its tasks (i.e., update the stock), and therefore, the microservice composition will never continue. Note that, in this case, the event that was triggered before the deleted one ("Enough Stock") was generated by the affected microservice (*Inventory*). Thus, Rule #1 cannot be applied. To face this change, the *Inventory* microservice can be modified by deleting the Intermediate Catching Event that receives the event "Payment OK" in such a way it can update the stock at the same time the payment is processed. As happens with the previous rule, the application of Rule #2 produces that two microservices that initially performed some of their tasks in a sequential way (e.g. first *Inventory* update stock and then the payment is processed) result in performing these tasks in a parallel way (e.g., after the local change, both the update of the stock and the order payment are executed when the "Payment OK" event is triggered, see Fig. 9B). Thus, a manual confirmation by the business engineer and the Shipment developer is needed.

3.2 Deleting a throwing event with attached data

This change implies the removal of a BPMN element that sends an event with attached data, which is required by other microservices. Two adaptation rules are proposed to support this change. Rule #3 considers that the data was produced previously by another microservice and just propagated by the modified microservice. Rule #4 considers the data is newly introduced by the modified microservice, and it does not exist in previous events of the composition.

A representative example of the change supported by Rule #3 is removing the BPMN Message Intermediate Throwing Event "Customer Checked" of the *Customers* microservice (see Fig. 10A). Note that we consider that this event carries the data of the purchase that is required by the *Inventory* microservice and that was initially introduced

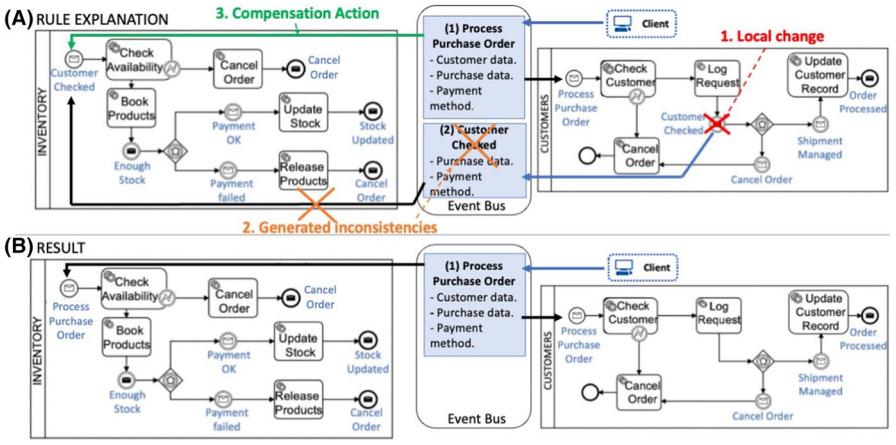

Fig. 10 Example of Adaptation Rule #3

in the composition by the client application. To allow the *Inventory* microservice to perform its tasks and maintain its participation in the composition, it can be modified to wait for an event that is triggered previously in the composition, and that contains the data that the *Inventory* microservice needs. In particular, the *Inventory* microservice can be modified to wait for the previous "Process Purchase Order" that also contains the required data. In this case, *Customers* and *Inventory* were initially executed in a sequential way, but after the modification (see Fig. 10B), they are executed in a parallel way because both are executed when the "Process Purchase Order" event is triggered. Thus, a manual confirmation by the business engineer and the Shipment developer is needed.

RULE #3

- **Affected element:** End Message Event or Intermediate Throwing Event.
- **Change:** Delete the affected element.
- **Conditions:**
  - The deleted event attaches some data (Rule #1 or #2 are applied otherwise).
  - The data is propagated and it is not introduced by the modified microservice (Rule #4 is applied otherwise).
- **Affected microservice(s):** Those that have a catching event waiting for the message sent by the deleted element.
- **Generated inconsistency:** Affected microservices will never start since their execution depends on the data attached to the event that is just deleted.
- **Compensation actions:** Modify the affected microservices to obtain the required data from a previous event.
- **Impact of the application:** Functional requirements are maintained. Coordination requirements are altered (the tasks of some microservices are performed in a different order as they were initially defined).
- **Adaptation type:** Automatic adaptation with acceptance.

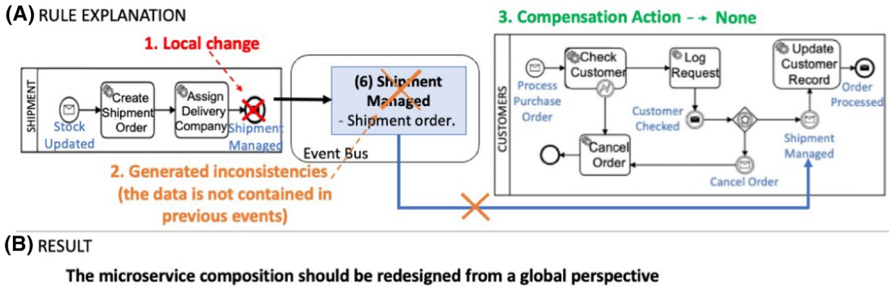

Fig. 11 Example of Adaptation Rule #4

RULE #4

- **Affected element:** End Message Event or Intermediate Throwing Event.
- **Change:** Delete the affected element.
- **Conditions:**
  - The deleted event attaches some data (Rule #1 or #2 are applied instead).
  - The data is newly created by the modified microservice (Rule #3 is applied otherwise).
- **Affected microservice(s):** Those that have a catching event waiting for the message sent by the deleted element.
- **Generated inconsistency:** Affected microservices will never start since their execution depends on the data attached to the event that is just deleted.
- **Compensation actions:** No compensation actions can be made in the affected microservices to obtain the required data.
- **Impact of the application:** Functional and coordination requirements cannot be maintained.
- **Adaptation type:** Global adaptation.

A representative example of the change supported by Rule #4 is removing the BPMN End Throwing Event "Shipment Managed" of the *Shipment* microservice (see Fig. 11A). Note that we consider that this event provides data about the shipment order (i.e., shipment company, delivery date, etc.), which the *Customers* microservice uses to update the customer record. If this event is not sent, the *Customers* microservice cannot continue its execution. In this case, there are no other events that provide specifically this data. Thus, to allow the *Customers* microservice to perform its tasks and maintain its participation in the composition, it is required to re-design the composition from a global perspective.

3.3 Updating a catching event with attached data

This change implies updating a BPMN element that defines the event that a microservice must listen at to execute some tasks. The event contains data that the microservice needs to complete its tasks.

Two scenarios are identified in this type of modification:

- Scenario A: The modified microservice is updated to catch an event that is not triggered in the context of the composition. Thus, the updated microservice will

no longer participate in the composition. The new event contains the data that the modified microservice requires to complete its process.

- Scenario B: The modified microservice is updated to catch another event that is already triggered within the composition. In this scenario, it is considered that developers update the catch element to expressly receive a new event that contains the data the modified microservice needs. Thus, the participation of the modified microservice will continue without generating any inconsistencies. Consequently, it is not necessary to apply any rule. Although, in this scenario, coordination requirements may change.

We propose Rule #15 to support scenario A.

RULE #15

- **Affected element:** Start Message Event or Intermediate Catching Event.
- **Change:** Update the data of the affected element.
- **Conditions:**
  - The updated event attaches some data (Rule #13 or #14 is applied otherwise[1])
  - The updated event contains at least the data required by the modified microservice.
  - There is at least one microservice in the composition that can send the updated event (Rule #16 is applied otherwise).
- **Affected microservice(s):** The modified microservice that has a catching event waiting for an event that is not being sent in the context of the composition.
- **Generated inconsistency:** The modified microservice will never start since its execution depends on the triggering of the new version of the updated event.
- **Compensation actions:** Search for one microservice that can send the updated event with the required data by the modified microservice and modify it to send the updated event.
- **Impact of the application:** Functional requirements are maintained but coordination requirements may change depending on the microservice modified to send the updated event.
- **Adaptation type:** Automatic adaptation with acceptance.

A representative example of the change supported by Rule #15 in scenario A is updating the BPMN Message Start Catching Event "Customer Checked" of the Inventory microservice. In this example, the Inventory microservice is modified to listen to a new event called "VIP Customer", and this new event should contain the purchase data, the payment method used by the customer, and the VIP discount of the customer (see Fig. 12). This new event does not exist in the composition since it is not triggered by any microservice. Therefore, the Inventory microservice will never start and the microservice composition is stopped. To solve this situation, we can modify the Customer microservice to send this new event instead of the "Customer Checked" event. Note that the Customers microservice can include the purchase data and the payment method in the new event "VIP Customer" since this data was already included in the old "Customer Checked" event. Regarding the VIP discount, the Customers microservice also has this data available since it is included in the customer data received in its starting event (see the Process Purchase Order event in Fig. 10). In this example, the coordination between microservices does not change, but depending on the affected microservice that is updated to send the new event, the coordination can change. Therefore, a manual confirmation by the business engineer and the Customers microservice

[1] These rules are not presented in this paper but can be found in [23].

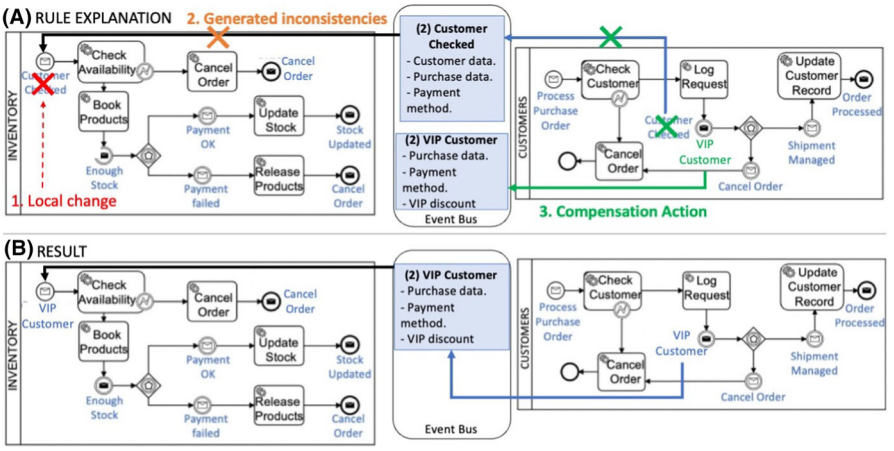

Fig. 12 Example of Adaptation Rule #15

developer is needed. Note also that another microservice could be waiting for the "Customer Checked" event that has been replaced by the "VIP Customer" event. In this case, the rule that faces the modification of an intermediate throwing event with data (Rule 11 or 12, see [23]) would be applied.

3.4 Further analysis of adaptation automation

This work faces the challenge of evolving microservice composition when a change is introduced from the local perspective of one partner. Microservice compositions are implemented as an event-based choreography of BPMN fragments, and they are deployed into an architecture in which they coexist with the big picture of the composition that is defined in a global BPMN model.

This section has introduced a catalogue of adaptation rules to automate, as much as possible, the adaptation of a microservice composition when a partner introduces a local change in its BPMN fragment. These rules focus on managing local changes that impact the coordination requirements of the composition since they need to be integrated with both the big picture of the composition and the rest of the microservices that participate in the choreography. Note that local changes that only impact functional requirements can be integrated into the big picture without integration with the rest of the partners. Also, note that: (1) a change that impacts the coordination requirements of an event-based choreography of BPMN fragments implies modifications to the events that the fragments either receive through BPMN catching events or send through BPMN throwing events; and (2) modification to these events can affect either the name of the event that is being sent/received or the data that optionally can be attached to it.

We have defined 19 adaptation rules that face 19 different local changes. They are classified into one of the following three categories depending on the degree of automation that each rule is able to achieve: Automatic adaptation, Automatic adap-

tation with acceptance, and Global adaptation. 3 out of these 19 rules were classified as Automatic adaptation in such a way they can adapt a composition to face a local change fully automatically. These rules face local changes that only affect the name of the interchanged events. Thus, when a microservice developer changes the name of an event of its BPMN fragment the whole microservice composition can be automatically adapted to face this change. 11 out of the 19 rules are classified as Automatic with acceptance in such a way they can automatically find a solution to adapt the event-based choreography and maintain the functional integrity of the microservice composition (i.e., all the functional requirements are satisfied). However, this solution implies some alteration in the choreography's flow (e.g., some microservices that were initially defined in a sequential way are executed in parallel after the application of the adaptation rule). Thus, a manual acceptance by the business engineer and the affected microservice developers is needed. These rules face changes that affect some data included in the events interchanged by the microservices. However, the affected data is available in other events triggered in the context of the choreography. Finally, 5 out of the 19 adaptation rules are classified as Global adaptation in such a way the impact of the local change on the event-based choreography is so significant that a solution to adapt the microservice composition cannot be automatically suggested. In these cases, the local changes identified by the adaptation rules imply the modification of events to include new data that cannot be found in other events of the choreography.

Thus, the proposed catalogue of adaptation rules can automatically suggest an adaptation in 14 out of the 19 identified local changes, where 11 of them require manual acceptance. In this sense, although a high degree of automation has been achieved, there is still space for improvement. We plan to study the integration of machine learning techniques into the proposed protocol in such a way we can reduce the adaptations that require manual acceptance or even those that have been classified as Global adaptation. To do so, the analysis of execution logs in addition to the characterization of the local change is being considered.

Finally, it is worth remarking that the 19 adaptation rules have been defined and validated with the development of different case studies. However, there is still the need to formally validate its completeness and correctness in such a way we can guarantee both (1) that all the local changes that can be done in an event-based choreography of BPMN fragments are supported by the adaptation rules; and (2) that every change is correctly faced by the proposed adaptation rules.

4 Proof-of-concept prototype

The evolution protocol for microservice composition that is presented in this paper has been implemented in the context of the microservice architecture[2] introduced in Sect. 1.1, which is presented in detail in [5]. We have extended the software infrastructure that supports this architecture with the following components (see Fig. 13):

[2] Specific tool support is available at: https://github.com/pvalderas/microservices-composition-infrastructure.

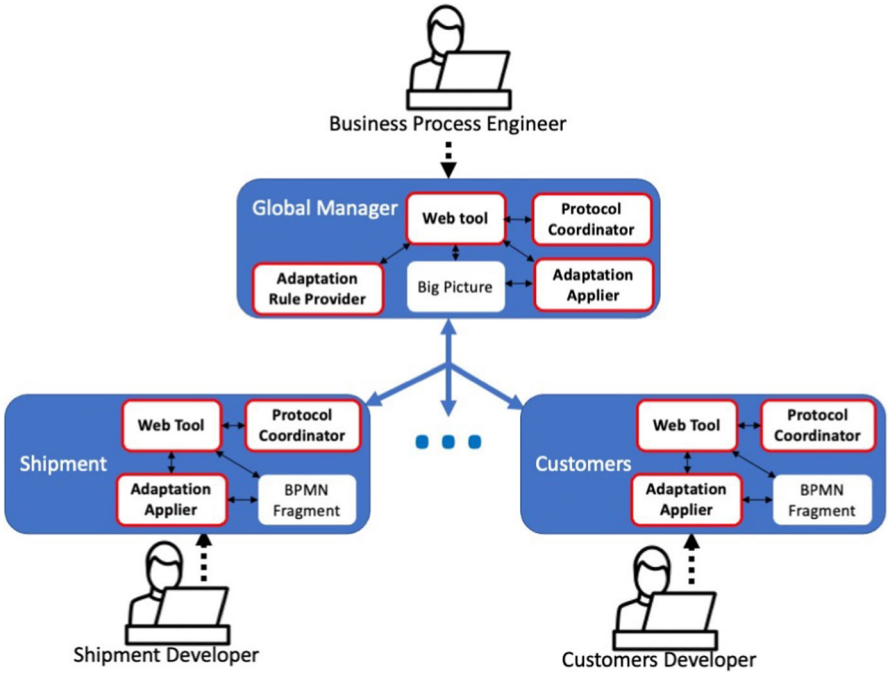

Fig. 13 Architecture extensions to support the evolution protocol

1. **Protocol coordination:** Both the Global Manger microservice and the business microservices have been extended with a software module that oversees the application of the different steps of the protocol depending on the change done and the required adaptation. This module also manages the interchange of messages among microservices required to synchronize the changes done in the composition.
2. **Adaptation Rule Provider:** The *Global Manager* microservice has been extended with a software module that contains the catalogue of adaptation rules and proposes the required rule to support a specific change.
3. **Adaptation Applier:** Both the *Global Manager* microservice and the business microservices have been extended with a software module that is in charge of applying adaptation rules in the big picture or the BPMN fragments either automatically, when possible, or after the manual acceptance by the business manager and microservice developers.
4. **Web Tool:** Both the *Global Manager* microservice and the business microservices have been extended with a web tool that provides business engineers and microservice developers with an interface that helps them analyse the modification scenario in order to take decisions when needed.

4.1 Realization details

This section presents some details of a realization of the architectural solution presented above as a prototype involving mapping technology choices into the solution concepts.

Microservices implementation and composition execution

1. Microservices have been implemented by using Java/Spring technology.
2. In order to allow each business microservice (e.g., Shipment, Customers) to execute BPMN fragments we have chosen the option of including a lightweight version of the Camunda BPMN engine[3] in each of them.
3. The web tool incorporated into both, the *Global Manager* microservice and the business microservices, is based on the bpmn.io[4] open-source tool, which is also supported by the Camunda project.
4. A RabbitMQ[5] message broker is used to implement the event bus. Topic-based queues are used to publish messages of one composition. Messages are implemented in JSON format and include a client ID to inform about the process instance they belong to.

Protocol communication issues

1. Each interaction among software participants during the application of the protocol (see Sect. 2) results in an interchange of a JSON message. The following is a representative example of a *coordChange* message sent to the Global Manager in the first phase of the protocol by a modified microservice (see Fig. 4):

```
{
  "modificationId": "284gf04y8359",
  "changeType": "coordination",
  "composition":"purchaseProcess",
  "microservice": "customers",
  "actions": [{
      "action":"deleteElement",
      "elementType":"IntermediateCatchEvent",
      "elementId":"IntermediateCatchEvent_06f6r4x",
      "name":"Customer Checked"
  }]
}
```

2. Messages are managed in an asynchronous way by a RabbitMQ message broker. Topic-based queues are used to coordinate the interchange of messages among software participants. For instance, there is a *localChanges* topic to which the Global Manager microservice is subscribed to receive the messages that any modified microservices publish with this topic.

[3] https://github.com/camunda/camunda-bpm-platform.

[4] https://github.com/bpmn-io.

[5] https://www.rabbitmq.com/.

Local adaptation issues

1. The local adaptation of a BPMN fragment can be considered as an endogenous model transformation [24], i.e., a transformation between two models (the original BPMN fragment and the adapted one) expressed in the same language. Currently, there are several solutions to implement model transformations[25]. In this work, we have used a direct manipulation approach based on the Java parser provided by the Camunda platform[6]. We have selected this option since it can be supported with other Java tools that facilitate the integration of the adaptation rules with the microservice architecture.
2. When BPMN fragments are adapted and deployed into the Camunda engine of business microservices, existing instances that run on previous versions are not affected. By default, Camunda implements a versioning strategy through which the engine checks if the version has changed when a process (in our case a BPMN fragment) with an existing ID is deployed. If it has, it will register that deployment as a new version of the process. Running instances will continue to run on the basis of the version they started with, new instances will be created based on the latest version of that process. However, Camunda provides a process instance migration API that can be employed to evolve running instances of a process when it is evolved. We plan to use this API in further work to support the evolution of running BPMN fragments instances.

4.2 Example of protocol application

As a representative example[7], we present some snapshots that illustrate how the protocol is supported by the developed tools in order to apply Rule #1 to face the deletion of a throwing event that sends a message without attached data (see Sect. 3.1).

As we can see in the representative application example presented below (see Figs. 14, 15, and 16), the web tool provides business process engineers and developers with a BPMN editor that graphically shows, in the same model, the change done to the microservice composition as well as the result of applying the proposed adaptation rules. To do so, a codification based on colours is used. In particular, deleted elements by a change done by a professional or as a result of the application of an adaptation rule are shown in red; elements that have been added are shown in green; and elements that have been modified are shown in orange. The main goal of this solution is to help professionals to take more informed decisions about the modification scenario with a graphical representation that include the key factors of it, i.e., the change done, and, if possible, the adaptation required and proposed by the system.

4.2.1 Phase 1: classification of the local change

During the first step the developer of the *Customers* microservice deletes the throwing event that sends the Customer Checked event in its BPMN fragment.

[6] https://github.com/camunda/camunda-bpm-platform/tree/master/model-api/bpmn-model.

[7] A video demo of this change can be found at: https://media.upv.es/#/portal/video/b051ddb0-9fcd-11ec-91a8-9b64e45e71c9.

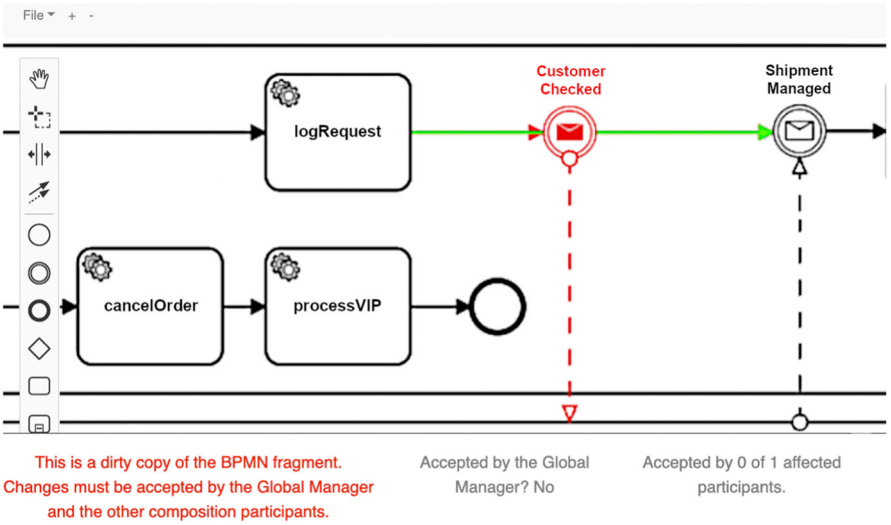

Fig. 14 The locally modified BPMN fragment is marked as dirty

As the change deletes a BPMN element that affects the coordination requirements, the web tool of the *Customers* microservice classifies it as a coordination change and sent it to the *Global Manager*. At this point, the *Customers* microservice needs to receive acceptance by the *Global Manager* and the affected microservices, which, in this case, is the *Inventory* microservice. In the meantime, the web tool marks the BPMN fragment as dirty, meaning that it cannot be modified. In addition, the web tool shows the dirty copy of a BPMN fragment and highlights the changes done in the corresponding colours (see Fig. 14) as follows, the elements deleted in the change are shown in red while those added are shown in green. Note also how the bottom side of the web tool informs the user that the BPMN fragment shown corresponds to a dirty copy of the fragment as well as whether or not it has been accepted by the Global Manager and the affected participants.

4.2.2 Phase 2: propagation of a coordination change

The *Global Manager* microservice receives the coordination change done by the *Customers* microservice. In this case, it detects that the change implies the deletion of an event-based communication element. Thus, it continues with the actions of Phase 3.

4.2.3 Phase 3: suggestion and realization of an adaptation

Considering the catalogue of adaptation rules, The *Global Manager* analyses the big picture of the microservice composition and identifies that the change affects the BPMN fragment of the *Inventory* microservice. Then, the *Global Manager* applies Rule #1 to adapt the *Inventory* BPMN Fragment. This adaptation is classified as an *Automatic adaptation with acceptance*.

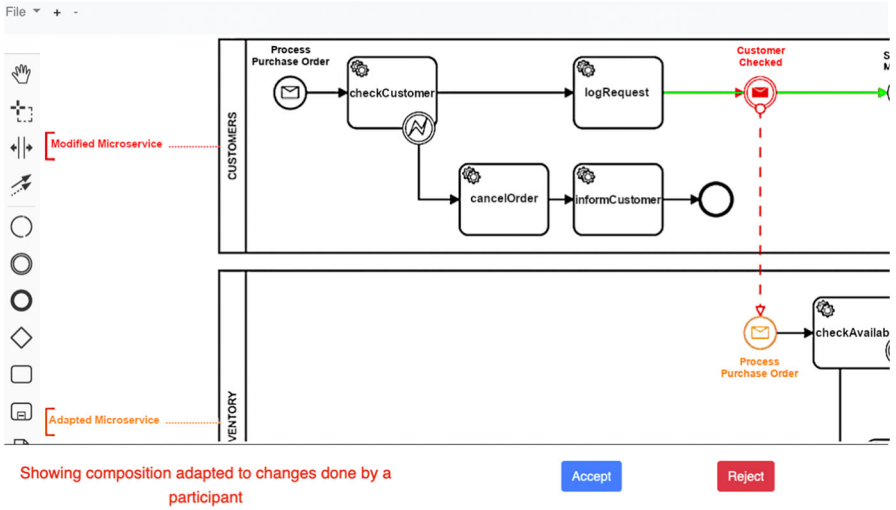

Fig. 15 Web tool of the Global Manager showing a local change and a suggested adaptation

At this point, the suggested adaptation must be manually accepted by the business engineer. To do so, the web tool of the *Global Manager* shows the big picture of the microservice composition with the change done by the developer of the *Customers* microservice and the adaptation proposed for the *Inventory* microservice (see Fig. 15). To do so, the web tool shows the involved changes in different colours. In particular, the red colour is used to mark the deleted elements, the green colour is used for those elements added in the change or the adaptation, and orange is used to highlight the updated ones. In addition, the web tool also includes some labels in order to mark the pool of the modified microservice (*Customers*) and the pool of the affected ones (*Inventory*). The web tool provides two buttons to allow the business engineer to either accept or reject the suggested adaptation. In this step, the business engineer could manually modify the suggested adaptation previously to accept it.

In this example, we consider that the BP engineer accepts the suggested adaptation. Then, the changes proposed for the *Inventory* microservice need to be accepted by its corresponding developer. The web tool of this microservice shows the adaption as it is illustrated in Fig. 16. As happened with the previous cases, changes are highlighted following the proposed codification of colours. In this case, we can see that the update of a catching event is depicted in orange.

In order to not overload this section we suppose that the microservice developer accepts the suggested adaptation. Then, it is not needed to start the negotiation phase (whose supporting web interfaces are the same as the ones shown above). Thus, the *Global Manager* is informed about the acceptance of the adaptation, and the change is integrated into the big picture. Afterwards, the *Customers* microservice can confirm the change and the *Inventory* microservice can confirm the adaptation.

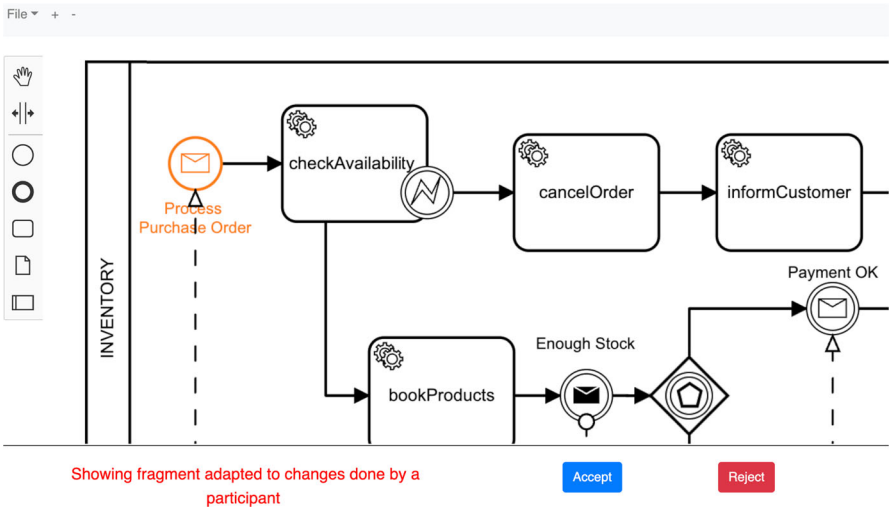

Fig. 16 Web tool of the *Inventory* microservice showing a suggested adaptation

5 Evaluation

In this section, we evaluate the proposed protocol and the supporting tools to evolve a microservice composition. Considering the research question stated in Sect. 1.3, the hypothesis that we wanted to validate was the following:

> *The evolution protocol and the supporting web tools are effective to evolve a microservice composition based on the choreography of BPMN fragments from the local perspective of a microservice.*

To do so, we arranged a controlled subject-based experiment in which participants played two roles: (1) microservice developers, which were asked to update or adapt BPMN fragments; and (2) business process engineers, which were asked to manage a local change from the big picture perspective. All of them had to use the web tools presented in the previous section. We set up the microservice architecture and deployed the microservice composition of the running example in such a way the big picture was available from the *Global Manager* microservices and each business microservice had its corresponding fragment.

This experiment was done by following the research methodology practices provided by [20]. Next, we introduce the experiment by describing its participants, design, execution, analysis of the results, and threats of validity.

5.1 Participants

A total of 12 subjects between 24 and 45 years old participated in the experiment (five female and seven male). Two participants worked for external computer science companies; five of them belonged to our research institute; and the remaining five

participants were doctoral students of the Universitat Politècnica de València. All participants had some experience in the modelling of BP with BPMN and only three participants had expertise in microservices.

5.2 Design

We arranged an experiment in which participants were grouped into four groups of three people each one and proposed several scenarios of microservice local modification. For each scenario, each member of a group must play a different role in such a way all the participants played the role of business engineer, developer of a modified microservice, and developer of an affected microservice. Note that we propose to participants a type of change to be done but we do not indicate to them which change exactly must be done. In the same way, the rest of the decisions were freely taken by participants in such a way changes were rejected or accepted according to their criteria. This allows us to evaluate if the developed tools provide the proper support to take these decisions. The instruments that were used to carry out the experiment were:

- **A demographic questionnaire:** it was used to know the level of the users' experience in process modelling, BPMN, and microservices.
- **Work description:** the description of the work that the subjects should carry out in the experiment. Each group oversaw performing these three local changes:
  - Perform a modification in functional requirements, i.e., add, delete, or update some BPMN task
  - Perform the following modifications in coordination requirements: (1) Add a Catch or Throwing Event to a microservice's fragment; (2) Update a Catch or Throwing Event in a microservice's fragment; (3) Delete a Catch or Throwing Event in a microservice's fragment.
- **A NASA-TLX questionnaire:** it was used to evaluate the perceived mental/physical/temporal demand, performance, effort, and frustration on a 100-point scale with 5-point steps. This questionnaire was extended with additional questions to ask some questions about the performed tasks and allow participants to introduce additional comments.

5.3 Execution

To perform the experiment, we organized a workshop with two sessions of three and four hours. In the first session, participants were asked to fill in the demographic questionnaire to capture their background and were trained in our microservice composition approach. In the second session, participants were distributed in groups and each of them was initially assigned a role (i.e., BP engineer, developer of the modified microservice, or developer of an affected microservice). We presented the motivating example and participants were invited to perform the local changes introduced above. Note that different changes were proposed in such a way we could evaluate the application of the protocol in different scenarios. After participants completed the

Table 1 Summary of the changes done in the experiment

	Modify a functional requirement	Add an event	Update an event	Delete an event
Applied automatically	12	12	4	–
Manual acceptance	–	–	5	7
Global redesign	–	–	3	5

proposed changes, each participant had to fill in the NASA-TLX questionnaire, indicating the role they played. Afterwards, they performed again the changes included in the task by changing their roles. In total, each change was performed 3 times per group. Throughout the second session, we observed participants and took notes on their behaviour.

5.4 Analysis of the results

Considering that each change was done three times per group, a summary of the changes done is shown in Table 1. As we can see, all the modifications to functional requirements and the creation of new coordination possibilities (i.e., the creation of new BPMN events) were automatically managed by the tool-supported protocol. This means that these changes were automatically synchronized with the big picture and the BPMN fragments of the other participants. Regarding the changes that imply updating a BPMN event, four of them were automatically managed since participants only update the events' names. The other eight changes introduced updates in the data published by the events. Five of them could be applied after a manual acceptance by the BP engineer and the microservice developers while two of them required a global redesign since changes in data were too significant. Finally, seven of the changes in which an event was deleted could be managed after a manual acceptance by the BP engineer and the microservice developers. In these cases, the event deleted did not have attached data or the attached data could be extracted from a previous event. In two of them, a negotiation between both professionals was performed. In the rest, developers of the affected microservices directly accept or reject the adaptation suggested by the BP engineer. The other five cases in which an event was deleted were classified to be globally redesigned since they significantly impact that data interchanged among microservices.

All the changes performed by participants could be successfully managed by the defined protocol and the supporting tools. The changes were properly identified by the new software components and the proposed adaptation rules were appropriate according to the analysis done in our catalogue [23]. The results obtained from the NASA-TLX questionnaires (average (Avg), median (Med), standard deviation (SD), best result (Best), and worst result (Worst) columns) as presented in Table 2. Figure 17 shows three box-and-whisker graphics of these results. As introduced above, the NASA-TLX questionnaire evaluates the perceived mental load (ML), physical demand (PD) temporal demand (TD), performance (P), effort (E), and frustration (F)

Table 2 NASA-TLX results

	Avg	Med	SD	Best	Worst
ML	29,2/14,2/24,2	30,0/15,0/25,0	6,7/3,6/5,1	15,0/10,0/15,0	40,0/20,0/35,0
PD	3,3/2,5/2,5	5,0/2,5/2,5	2,5/2,6/2,6	0,0/0,0/0,0	5,0/5,0/5,0
TD	26,3/17,1/14,2	22,5/17,5/15,0	8,0/6,2/3,6	20,0/10,0/10,0	45,0/25,0/20,0
P	18,3/17,9/24,2	15,0/17,5/25,0	4,9/3,3/4,7	15,0/15,0/20,0	30,0/25,0/35,0.
E	19,2/18,3/13,3	20,0/20,0/12,5	6,0/5,4/3,9	10,0/10,0/10,0	30,0/25,0/20,0.
F	32,1/15,0/41,3	35,0/15,0/37,5	10,1/6,0/11,5	15,0/5,0/25,0	45,0/25,0/60,0

These values correspond to the BP Engineer/Developer of Modified Microservices/Developer of Affected Microservices

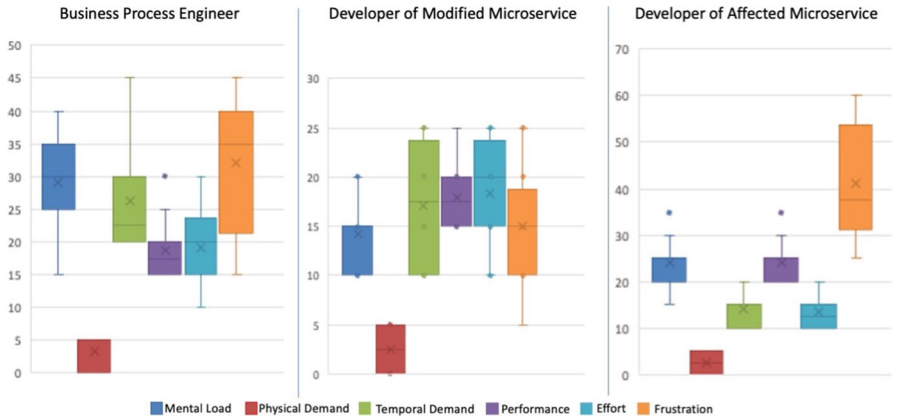

Fig. 17 Box-and-whisker graphics of the NASA-TLX results

on a 100-point scale, where the highest scores represent the worst results. Thus, mental/physical/temporal demand, effort and frustration are rated between very low (value 0) and very high (value 100); while performance is rated between very good (value 0) and very bad (value 100). We present the value obtained for the tasks that participants did by playing each role: business process manager, developer of a modified microservice, and developer of an affected microservice.

The tasks that the participants did playing the role of a BP engineer were the ones with the highest mental load and temporal demand. This is an expected result since BP engineers are those that must analyse the big picture of the composition to accept or reject the adaptation rules when manual acceptance is needed. Despite this, the obtained values are quite good. The performance has been evaluated with a good mark, which is reinforced by the comments of participants that felt the tools helped them to analyse the changes done and the impact on the global composition. They also found useful the automatic proposition of a possible adaptation when the change cannot be managed automatically. Another aspect to be highlighted is that frustration is higher than the other measures. The main reasons given by participants for this evaluation were two. On the one hand, the web tool depicted some elements overlapped when showing the proposed adaptation in the BPMN model with the big

picture. For instance, if a BPMN element was replaced by another one, the tool showed the deleted element in red overlapped by the new element in green (an example of this issue can be seen in Fig. 15). In these cases, participants had to move elements to properly see them. This is a graphical visualization problem that we plan to improve in further versions of the developed tools. On the other hand, it was also difficult for them to identify the adaptation proposed for each change when two or more changes were done at the same time. For instance, some participants deleted more than one element when we asked them to delete a BPMN event. In these cases, the tool showed all the changes performed (i.e., the deleted events) and the proposed adaptations for each of them. However, the tool did not inform about which adaptation corresponded to each change. This was a little confusing for participants. To improve this problem, we want to include additional annotations in the BPMN model of the big picture that provides additional information about the applied rules and the change they are supporting.

As for the tasks of the developer of a modified microservice, they obtained the best values in this evaluation and there are few aspects to highlight. Participants just need to do a change and wait for the acceptance or rejection of the BP engineer and the developers of the affected microservices. Participants found the tool and the supported protocol easy to use and intuitive. One improvement that we identified from their comments was the inclusion of an explanation of why a change is rejected. In the current version of the tools, developers of a modified microservice are informed about the rejection of a change, and it is rollbacked in its BPMN fragment. However, a reason for this rejection is not given. A change can be automatically rejected because the proposed adaptation rule is classified as a *Global adaptation*. But a change can also be manually rejected by the BP engineer or the developer of an affected microservice after analysing its impact. We are studying to include a textual description that informs about how a change has been rejected (i.e., automatically or manually) as well as to extend the web tools in order to allow the BP engineer and the developers of the affected microservice to introduce a reason for the rejection.

Although not directly related to the proposed protocol, we detected a problem when participants changed their BPMN fragments: some of them introduced syntactical mistakes in the model (e.g., a missed flow sequence that left an unconnected element) that were propagated to the big picture. A solution to this problem could be adapting the tools in order to allow modifying models only through the applications of change patterns [26], guaranteeing that models are syntactically correct by construction.

Finally, as for the tasks of the developer of an affected microservice, participants found the tool-supported protocol useful and effective to manage local changes. However, they felt a little confused in some modification scenarios, which is the reason for being the task with the highest frustration. In scenarios such as the one presented in Sect. 4.1, the proposed adaptation implies changing the event that triggers the BPMN fragment of a microservice by another event (e.g., in the *Inventory* microservice, the Customer Checked event is changed by the Process Purchase Order, see Fig. 16). This adaptation was classified as automatic with acceptance because some tasks changed from a sequential execution to a parallel one. Thus, the developer of the affected microservice was requested to accept this change and was informed about the change of event but they were not informed about the reason. Some participants suggested changing how to address these scenarios by being manually accepted by the BP engi-

neer and automatically accepted by the affected microservices. We want to study the problem in detail in order to decide on applying the suggested solution or introduce additional information for the developers of the affected participants and maintain the two-steps manual acceptance.

5.5 Conclusions

In all the modification scenarios faced during the experiment, participants could achieve a resolution in order to either accept and integrate the local change or reject it. In addition, comments given by participants and the results obtained in the NASA-TLX questionnaire reinforce the potential usefulness of the approach. In this sense, we can accept the validation hypothesis and conclude that the proposed evolution protocol and the supporting tools are effective enough to evolve a microservice composition based on the choreography of BPMN fragments from the local perspective of a microservice.

Of course, the experiment has also allowed us to detect some usability problems in the proposed tools, which should be solved to facilitate the adoption of the whole approach in order to create microservice compositions and manage further evolutions. These problems have been precisely identified and delimited. Solutions to improve them have been stated and will be considered in future work.

5.6 Threats to validity

According to the classification of threats to validity presented in [27], the threats that are applicable to our evaluation are the following:

Conclusion validity. This experiment was threatened by the random heterogeneity of subjects, which was minimized with: (1) the demographic questionnaire that allowed us to evaluate the knowledge and experience of each participant beforehand; and (2) the training sessions in which all subjects participated to have a similar background in our proposed microservice composition approach.

Construct validity. This experiment was threatened by the threat of hypothesis guessing (people might try to figure out what the purpose and intended result of the experiment are), which was minimized by hiding the goal of the experiment (i.e., which was the validation hypothesis).

Internal validity. This experiment was threatened by the diffusion or imitation of treatment, which occurs when some participant learns from the experience of others. We reduced this treatment by making all the groups perform the experiment at the same time and physically separating them to avoid interactions. We also observe the behaviour of each participant in order to guarantee isolated work as much as possible.

External validity. This type of validity concern is related to conditions that may limit our ability to generalize the results of the experiment. Just one case study was used in the experiment, which can threaten the generalizability of this experiment. Also, most of the participants were from the academic environment, which could threaten the generalization to another population. Thus, usability experiments with additional case studies and different participant profiles are needed.

6 Related work

There are several works that face the problem of defining a composition of microservices at a high level of abstraction [28–32], or from an architectural point of view [33–35]. However, none of these works considers the evolution of microservice compositions once they are deployed into a system. Similarly, in web service compositions, [36] proposes an abstract model defined in UML that can be used to build a choreography of services from two perspectives as we do in our approach: top-down and bottom-up. However, this solution does not include mechanisms to support the evolution of the proposed model either.

In the literature we find many works addressing the problems and challenges that involve the evolution of compositions in the context of services, microservices and web services either at design time or at runtime. However, the objectives sought, and the techniques used vary from one proposal to another as it is explained next.

Regarding the works that face the evolution at design time, i.e., focusing on the propagation of changes over the composition model without considering runtime adaptations for the affected participants, we find, within the context of microservice compositions, [37] which proposes a UML model that is based on dividing the architecture of an application into three layers: the architecture layer, the instance layer, and the infrastructure layer. The model is built using information from system logs, infrastructure data, messages, and inter-service operations. Based on this model, this work focuses on evolving the system in terms of the required number of microservices, by proposing the creation of new ones or the removal of existing ones. However, it does not address how to propagate the changes that occur in a microservice to the global composition. Cornax et al. [38] proposes the use of a UML sequence diagram to represent a choreography and a refinement process to obtain a definition of this choreography based on the open-source choreography programming language AIOCJ. This work allows the evolution of the microservice composition from a top-down perspective. However, a bottom-up evolution that allows changes from the local perspective of a microservice is not supported. Giallorenzo et al. [39] presents a model that extends BPEL4WS to automate the dynamic linking of web services in the context of a composition. The work proposes the creation of a service that integrates new services into the composition, following a bottom-up strategy. This solution allows the evolution of the composition from a bottom-up perspective. However, a composition is implemented as a centralized orchestration. We implement microservices composition through distributed choreographies, which reinforce the independence among services that is demanded in microservice architectures. Fdhila et al. [40, 41] propose models to describe web service orchestrations and fragment them into distributed choreographies. Then, the evolution is faced from the orchestration model and propagated to the model fragments. However, the integration of a change directly introduced in a fragment is not considered. With regards solutions that propagate changes in a choreography, [42] presents an approach to propagate changes among partners but adaptations to the affected partners are not provided. Fdhila et al. [43] also faces the challenge of propagating changes in decentralized choreographies. However, they only consider changes in the public interface of participants. We go a step further trying to adapt the internal model of participants. Weidlich et al. [44]

proposes a solution to synchronize aligned models when one of them changes; [45] allows choreography evolution by postponing decisions later in the lifecycle through abstract model constructions; [46] allows reconfiguring the choreography but saving the results already obtained in the execution. This is achieved by defining an algorithm and a system for execution and monitoring of choreography instances. Wombacher [47] considers two different visions of BPEL processes, one defined as a choreography and another one as an orchestration, and presents an approach to propagate changes from the choreography level to the orchestration level. However, compared to our proposal, none of these works supports the introduction of local changes in one of the participants of the choreography and the propagation of the changes to the rest of the members.

Regarding the works that face the evolution at runtime, i.e., focusing on adapting the participants of the choreography to the introduction of changes, we find [48], which proposes a self-adaptive model that can evolve in runtime to solve the problem of the optimal size of granularity of microservices. The model is based on a MAPE-K loop to create a systematic solution that improves the lifecycle of a microservice by accumulating knowledge and establishing parameters to know when a microservice needs to be divided or merged. This work focuses on the decomposition of a system into microservices but pays little attention to their composition. Florio [49] proposes an infrastructure called GRU, which uses self-adaptive techniques based on agents to manage large-scale distributed systems. These works focus on the automatic adaptation of microservices to manage resource consumption, in such a way aspects such as scalability, fault tolerance and performance can be improved. Again, this work does not consider the composition of microservices. Kolb et al. [50] and Mafazi et al. [51] present two approaches to update process models from the changes done to personalized views on them. In these cases, evolution is faced from centralized models instead of a split and distributed model as we do in this work. Andrikopoulos et al. [52] applies evolution to handle errors and to refine activities using AI planning techniques. Képes et al. [53] optimizes the system by adapting it to the current context captured by IoT sensors, and automatically transforming independent workflows models into situation-aware workflow models that cope with dynamic contextual situations. Mahfouz et al. [54] agrees on customization alternatives to fulfil the business needs describing constraints that govern the behaviour of participants. In [55], the authors address flexibility in business processes within the context of batch processing. To this end, they introduce flexible batch activities by specifying three different strategies (modelling, deployment, and execution strategies) which allow adapting business processes during the respective phases of the process lifecycle. Compared to our proposal, none of these works supports the introduction of changes in one of the participants of the choreography and the adaptation of the rest of the participants if the integrity of the choreography is affected by the introduced change. Additionally, there are other areas of interest where adaptation processes are also proposed. In the area of swarm intelligence, in [56] and [57] the entities that compose complex systems are able to adapt their behaviour using simulators. In the area of configurable workflows, [58] proposes an adaptation process to change the appearance of applications without changing the underlying implementation by proposing a framework with a set of services to isolate the different components of the application. In the area of

Quality of System (QoS) optimization, [59] combines a set of techniques and algorithms to transform and optimize a composition in terms of QoS. Finally, in the area of collaborative BPs, we find [60] which presents a model for dynamic binding where process participants can collectively agree on how to steer a BP; [61] which exposes the problem of runtime adaptation under the assumption that unexpected situations can be characterized by contextual elements; and [62] which automates the construction of exclusive choices considering multiple paths under a set of specific variable conditions to reconfigure BPs at runtime. Our work differs from all these proposals in that we propose a protocol to integrate local modifications that can change the functionality and the communication of a participant in the composition process, and additionally, we also support the negotiations between different participants if the communication among microservices has changed to agree on the new workflow of the composition.

7 Conclusions and further work

This work has presented an approach to manage the evolution of microservice compositions from the local perspective of one participant. These compositions are supported by a microservice architecture in which coexists two descriptions of it: (1) a global BPMN model that maintains the big picture of a composition; and (2) a split version that distributes the microservice responsibilities through BPMN fragments that are used to implement the composition as an event-based choreography. Thus, the evolution of a microservice composition when a participant in the choreography introduces a local change implies the integration of the change, when possible, in the affected BPMN fragments as well as the big picture.

To support this type of evolution we have proposed a protocol to manage the propagation of a change to the affected participants, the proposal of an adaptation to its BPMN fragments in order to maintain the functional integrity of the choreography, and the possible negotiation of this adaptation. This protocol also considered the participation of the business process engineer to synchronize all the changes with the big picture. To automatically propose adaptations for the affected BPMN fragments, a catalogue of 19 adaptation rules has been proposed. These rules precisely characterize different types of changes in a BPMN fragment, determines the affected participants, and propose an adaptation to their respective fragments. Both the automatic suggestion of adaptation rules and the application of the protocol have been supported by specific tool support, which has been integrated into the microservice architecture that supported the composition of microservices by means of a choreography of BPMN fragments. The protocol and the supporting tool have been successfully validated in an experiment with users.

The approach proposed in our previous work to compose microservices through a choreography of BPMN fragments introduced two main benefits: (1) it facilitated business process engineers to analyse the control flow if the composition's requirements need to be modified; and (2) it provided a high level of decoupling in the execution of microservices making it easy that the development team of each microservice can manage its BPMN fragment independently from the others. In this work, we reinforce these two benefits by improving the evolution of these compositions. On the one hand,

a top-down evolution (i.e., from the big picture to the BPMN fragments) was natively supported by our previous work. On the other hand, our current proposal introduces the possibility of a bottom-up evolution of composition (i.e., from the local perspective of a microservice). In addition, note that developers can manage the evolution of a microservice composition through models of a high-level abstraction defined in BPMN instead of having to handle hard-coded implementations or complex formal specifications.

As ongoing work, we are currently working on improving the supporting tool presented in Sect. 4 by considering the problems detected in the experiment as well as the comments and suggestions did by the participants. In addition, as further work, we want to improve the case study presented in this work to encourage participants to make changes that lead to negotiations so that we can validate this phase of the protocol more precisely. In addition, we also plan to integrate the proposed protocol with machine learning techniques that improve the suggestion of adaptation rules to face the local change of microservices. To do so, we are validating the approach with additional case studies in order to prepare a dataset that can be used to train a machine-learning algorithm. As a first step, we want to train classification models such as Nearest Neighbours, Decision Trees or Naive Bayes that do not need big datasets to work properly.

Another important issue to be faced as further work is the definition of deadlock-free and fault-tolerant compositions. To face this challenge we plan to support each microservice with multiple instances supported by brokers such as Kafka or RabbitMQ, which can guarantee that each event is only processed by one instance. Also, we plan to study works such as [63] to extend our approach with the possibility of defining timeouts so business microservices have the capability to cancel a composition if a specific timeout is reach.

Finally, we also need to consider the implementation of behavioural consistency mechanisms to (1) ensure that an implemented microservice is compatible with the rest of the participants of the composition when a modification is introduced, and (2) avoid deadlocks and guarantee a proper termination of the composition process [64]. Also, local enforceability should be studied [65], since the global model, which represents the whole composition, may capture behavioural constraints that cannot be enforced locally and therefore, may not be translatable into the local fragments.

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.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Weske M (2019) Business process management - concepts, languages, architectures, third edition. Springer, Berlin Heidelberg, pp 1–417 . https://doi.org/10.1007/978-3-662-59432-2
2. Barros A, Hettel T, Flender C (2010) Process choreography modeling. Handbook on Business Process Management 1. Springer, Berlin Heidelberg, pp 257–277 https://doi.org/10.1007/978-3-642-00416-212
3. Lewis J, Fowler M (2014) Microservices. Last accessed 4 April 2022 . https://martinfowler.com/articles/microservices.html
4. Ciancia V, Ferrari G, Guanciale R, Strollo D (2010) Event based choreography. Sci Comput Program 75(10):848–878. https://doi.org/10.1016/j.scico.2010.02.009
5. Valderas P, Torres V, Pelechano V (2020) A microservice composition approach based on the choreography of bpmn fragments. Inf Softw Technol 127:106370. https://doi.org/10.1016/j.infsof.2020.106370
6. OMG: Business Process Model and Notation (BPMN), Version 2.0. Object Management Group. http://www.omg.org/spec/BPMN/2.0
7. van der Aalst WMP (2012) A decade of business process management conferences: Personal on a developing discipline. In: 10th international conference on business process management, Tallinn, Estonia, pp. 1–16. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32885-5_1
8. Casati F, Ceri S, Pernici B, Pozzi G (1998) Workflow evolution. Data Knowl Eng 24(3):211–238. https://doi.org/10.1016/S0169-023X(97)00033-5
9. Rinderle S, Reichert M, Dadam P (2004) Correctness criteria for dynamic changes in workflow systems - a survey. Data Knowl Eng 50(1):9–34. https://doi.org/10.1016/j.datak.2004.01.002
10. Lenz R, Reichert M (2007) IT support for healthcare processes - premises, challenges, perspectives. Data Knowl Eng 61(1):39–58. https://doi.org/10.1016/j.datak.2006.04.007
11. Mulyar N, van der Aalst WM, Russell N (2008) Process flexibility patterns. Technische Universiteit Eindhoven
12. Cognini R, Corradini F, Gnesi S, Polini A, Re B (2018) Business process flexibility-a systematic literature review with a software systems perspective. Inf Syst Front 20(2):343–371. https://doi.org/10.1007/s10796-016-9678-2
13. Reichert M, Weber B (2012) Flexibility issues in process-aware information systems. In: Enabling Flexibility in Process-Aware Information Systems. Springer, Berlin Heidelberg, pp 43–55
14. Ortiz-Amaya J, Torres Bosch MV, Valderas P (2020) Characterization of bottom-up microservice composition evolution. an approach based on the choreography of bpmn fragments. In: Conceptual modeling. 39th international conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings, pp. 101–114 . Springer Nature
15. Ortiz J, Torres V, Valderas P (2022) Supporting a bottom-up evolution of microservice compositions based on the choreography of bpmn fragments. In: Advances in information systems development. Springer, Berlin Heidelberg, pp 219–236
16. Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Q 28(1):75–105
17. Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S (2008) A design science research methodology for information systems research. J Manag Inf Syst 24(3):45–77
18. Avison DE, Lau FY, Myers MD, Nielsen PA (1999) Action research. Commun ACM 42(1):94–97. https://doi.org/10.1145/291469.291479
19. Larman C, Basili VR (2003) Iterative and incremental development: a brief history. Computer 36(6):47–56. https://doi.org/10.1109/MC.2003.1204375
20. Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164. https://doi.org/10.1007/s10664-008-9102-8
21. Butler D, Aspinall D, Gascón A (2019) On the formalisation of $\sum$-protocols and commitment schemes. In: Principles of security and trust. Springer, Cham, pp 175–196. https://doi.org/10.1007/978-3-030-17138-4_8
22. Skeen D (1981) Nonblocking commit protocols. In: Proceedings of the 1981 ACM SIGMOD international conference on management of data, Ann Arbor, Michigan, USA. ACM Press, USA, pp 133–142 . https://doi.org/10.1145/582318.582339

23. Ortiz Amaya J, Torres Bosch MV, Valderas Aranda PJ (2022) A catalogue of adaptation rules to support local changes in microservice compositions implemented as choreographies of bpmn fragments. Technical report, Universitat Politècnica de València . http://hdl.handle.net/10251/181551
24. Mens T, Van Gorp P (2006) A taxonomy of model transformation. Electron Notes Theor Comput Sci 152:125–142. https://doi.org/10.1016/j.entcs.2005.10.021
25. Czarnecki K, Helsen S (2003) Classification of model transformation approaches. In: Proceedings of the 2nd OOPSLA workshop on generative techniques in the context of the model driven architecture, vol. 45, pp. 1–17 . USA
26. Weber B, Reichert M, Rinderle-Ma S (2008) Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl Eng 66(3):438–466. https://doi.org/10.1016/j.datak.2008.05.001
27. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B (2012) Experimentation in software engineering. Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2
28. Gutiérrez-Fernández AM, Resinas M, Cortés AR (2016) Redefining a process engine as a microservice platform. In: Business process management workshops - BPM 2016 international workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers. Lecture notes in business information processing. vol 281, pp 252–263. https://doi.org/10.1007/978-3-319-58457-7_19
29. Petrasch R (2017) Model-based engineering for microservice architectures using enterprise integration patterns for inter-service communication. In: 14th International joint conference on computer science and software engineering (JCSSE), NakhonSiThammarat, Thailand, pp 1–4. IEEE, USA . https://doi.org/10.1109/JCSSE.2017.8025912
30. Guidi C, Lanese I, Mazzara M, Montesi F (2017) Microservices: a language-based approach. CoRR abs/1704.08073. https://doi.org/10.48550/arXiv.1704.08073
31. Brüggemann ME, Vallon R, Parlak A, Grechenig T (2014) Modelling microservices in email-marketing - concepts, implementation and experiences. In: 9th international conference on software paradigm trends (ICSOFT-PT), Vienna, Austria. SciTePress, USA, pp 67–71. https://doi.org/10.5220/0005000800670071
32. Safina L, Mazzara M, Montesi F, Rivera V (2016) Data-driven workflows for microservices: Genericity in jolie. In: 30th IEEE international conference on advanced information networking and applications (AINA) 2016, Crans-Montana, Switzerland. IEEE Computer Society, USA, pp 430–437. https://doi.org/10.1109/AINA.2016.95
33. Oberhauser R (2016) Microflows: Lightweight Automated Planning and Enactment of Workflows Comprising Semantically-Annotated Microservices. In: Proceedings of the sixth international symposium on business modeling and software design - BMSD, pp 134–143. https://doi.org/10.5220/0006223001340143. INSTICC
34. Ben Hadj Yahia E, Réveillère L, Bromberg Y, Chevalier R, Cadot A (2016) Medley: An event-driven lightweight platform for service composition. In: 16th international conference on web engineering (ICWE), Lugano, Switzerland. Lecture notes in computer science, vol 9671. Springer, Berlin, pp 3–20. https://doi.org/10.1007/978-3-319-38791-8_1
35. Monteiro D, Gadelha R, Maia PHM, Rocha LS, Mendonça NC (2018) Beethoven: an event-driven lightweight platform for microservice orchestration. In: 12th European conference on software architecture (ECSA), Madrid, Spain. Springer, Berlin, pp 191–199. https://doi.org/10.1007/978-3-030-00761-4_13
36. Mandell DJ, McIlraith SA (2003) Adapting BPEL4WS for the semantic web: the bottom-up approach to web service interoperation. In: Second international semantic web conference (ISWC), Sanibel Island, FL, USA. Lecture notes in computer science, vol 2870. Springer, Berlin, pp 227–241. https://doi.org/10.1007/978-3-540-39718-2_15
37. Sampaio AR, Kadiyala H, Hu B, Steinbacher J, Erwin T, Rosa NS, Beschastnikh I, Rubin J (2017) Supporting microservice evolution. In: International conference on software maintenance and evolution (ICSME), Shanghai, China. IEEE Computer Society, USA, pp 539–543. https://doi.org/10.1109/ICSME.2017.63
38. Cornax MC, Dupuy-Chessa S, Rieu D (2011) Bridging the gap between business processes and service composition through service choreographies. In: Engineering methods in the service-oriented context - IFIP WG 8.1 working conference on method engineering, ME, Paris, France. Springer, Berlin, pp 190–203. https://doi.org/10.1007/978-3-642-19997-4_18

39. Giallorenzo S, Lanese I, Russo D (2018) Chip: A choreographic integration process. In: Confederated international conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta. Springer, Berlin, pp 22–40. https://doi.org/10.1007/978-3-030-02671-4_2
40. Fdhila W, Rinderle-Ma S, Baouab A, Perrin O, Godart C (2012) On evolving partitioned web ervice orchestrations. In: 2012 Fifth IEEE international conference on service-oriented computing and applications (SOCA), Taipei, Taiwan, December 17-19, 2012. IEEE Computer Society, USA, pp 1–6. https://doi.org/10.1109/SOCA.2012.6449446
41. Fdhila W, Baouab A, Dahman K, Godart C, Perrin O, Charoy F (2011) Change propagation in decentralized composite web services. In: 7th International conference on collaborative computing: networking, applications and worksharing, CollaborateCom, Orlando, FL, USA, pp 508–511. ICST / IEEE, USA . https://doi.org/10.4108/icst.collaboratecom.2011.247097
42. Rinderle S, Wombacher A, Reichert M (2006) Evolution of process choreographies in DYCHOR. In: Confederated international conferences, CoopIS, DOA, GADA, and ODBASE, Montpellier, France. Springer, Berlin, pp 273–290. https://doi.org/10.1007/11914853_17
43. Fdhila W, Indiono C, Rinderle-Ma S, Reichert M (2015) Dealing with change in process choreographies: Design and implementation of propagation algorithms. Inf Syst 49:1–24. https://doi.org/10.1016/j.is.2014.10.004
44. Weidlich M, Mendling J, Weske M (2012) Propagating changes between aligned process models. J Syst Software 85(8):1885–1898. https://doi.org/10.1016/j.jss.2012.02.044
45. Weiß A, Sáez SG, Hahn M, Karastoyanova D (2014) Approach and refinement strategies for flexible choreography enactment. In: OTM Confederated International conferences "On the move to meaningful internet systems". vol 8841, pp 93–111. https://doi.org/10.1007/978-3-662-45563-0_6
46. Weiß A, Andrikopoulos V, Hahn M, Karastoyanova D (2020) Model-as-you-go for choreographies: rewinding and repeating scientific choreographies. IEEE Trans Serv Comput 13(5):901–914. https://doi.org/10.1109/TSC.2017.2732988
47. Wombacher A (2009) Alignment of choreography changes in bpel processes. In: 2009 IEEE international conference on services computing. IEEE, New York, pp 1–8 . https://doi.org/10.1109/SCC.2009.11
48. Hassan S, Bahsoon R (2016) Microservices and their design trade-offs: A self-adaptive roadmap. In: International conference on services computing, (SCC), San Francisco, CA, USA. IEEE Computer Society, USA, pp 813–818. https://doi.org/10.1109/SCC.2016.113
49. Florio L (2015) Decentralized self-adaptation in large-scale distributed systems. In: 10th joint meeting on foundations of software engineering (ESEC/FSE), Bergamo, Italy, pp. 1022–1025. ACM, USA . https://doi.org/10.1145/2786805.2803192
50. Kolb J, Kammerer K, Reichert M (2012) Updatable process views for user-centered adaption of large process models. In: 10th international conference (ICSOC), Shanghai, China. Springer, Berlin, pp 484–498. https://doi.org/10.1007/978-3-642-34321-6_32
51. Mafazi S, Grossmann G, Mayer W, Stumptner M (2013) On-the-fly change propagation for the co-evolution of business processes. In: Confederated international conferences: CoopIS, DOA-trusted cloud, and ODBASE, Graz, Austria. Springer, Berlin, pp 75–93. https://doi.org/10.1007/978-3-642-41030-7_6
52. Andrikopoulos V, Bucchiarone A, Saez SG, Karastoyanova D, Mezzina CA (2013) Towards modeling and execution of collective adaptive systems. In: Lomuscio A, Nepal S, Patrizi F, Benatallah B, Brandic I. (eds.) Service-oriented computing - ICSOC 2013 workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers. Lecture notes in computer science, vol 8377. Springer, Berlin Heidelberg, pp 69–81. https://doi.org/10.1007/978-3-319-06859-6_7
53. Képes K, Breitenbücher U, Sáez SG, Guth J, Leymann F, Wieland M (2016) Situation-aware execution and dynamic adaptation of traditional workflow models. In: Lecture notes in computer science, vol. 9846. Springer, Berlin Heidelberg, pp 69–83. https://doi.org/10.1007/978-3-319-44482-6_5
54. Mahfouz A, Barroca L, Laney R, Nuseibeh B (2009) Requirements-driven collaborative choreography customization. In: Service-oriented computing, vol 5900, pp 144–158. https://doi.org/10.1007/978-3-642-10383-4_10
55. Pufahl L, Karastoyanova D (2018) Enhancing Business Process Flexibility by Flexible Batch Processing. In: OTM confederated international conferences "On the move to meaningful internet systems", vol 11229, pp 426–444. https://doi.org/10.1007/978-3-030-02610-3_24

56. Pinciroli C, Trianni V, O'Grady R, Pini G, Brutschy A, Brambilla M, Mathews N, Ferrante E, Di Caro G, Ducatelle F, Birattari M, Gambardella LM, Dorigo M (2012) Argos: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intell 6(4):271–295. https://doi.org/10.1007/s11721-012-0072-5
57. Levi P, Kernbach S (2010) Symbiotic multi-robot organisms: reliability, adaptability, evolution, 1st ed. 2010. edn. Springer, Berlin, Heidelberg . https://doi.org/10.1007/978-3-642-11692-6
58. Chang JG, Sun W, Huang Y, Zhi HW, Gao B (2007) A framework for native multi-tenancy application development and management. In: Proceedings - the 9th IEEE international conference on E-commerce technology; the 4th IEEE international conference on enterprise computing, E-commerce and E-services, CEC/EEE 2007, pp 551–558. https://doi.org/10.1109/CEC-EEE.2007.4
59. Rosenberg F, Celikovic P, Michlmayr A, Leitner P, Dustdar S (2009) An end-to-end approach for qos-aware service composition. In: 2009 IEEE international enterprise distributed object computing conference, pp 151–160. IEEE, New York, NY, USA . https://doi.org/10.1109/EDOC.2009.14
60. López-Pintado O, Dumas M, García-Bañuelos L, Weber I (2022) Controlled flexibility in blockchain-based collaborative business processes. Inf Syst 104:101622. https://doi.org/10.1016/j.is.2020.101622
61. Nunes VT, Santoro FM, Werner CML, Ralha CG (2018) Real-time process adaptation: a context-aware replanning approach. IEEE transactions on systems, man, and cybernetics. Systems 48(1):99–118. https://doi.org/10.1109/TSMC.2016.2591538
62. Heinrich B, Klier M, Zimmermann S (2015) Automated planning of process models: design of a novel approach to construct exclusive choices. Decis Support Syst 78:1–14. https://doi.org/10.1016/j.dss.2015.07.005
63. Guermouche N, DalZilio S (2011) Formal requirement verification for timed choreographies. Int J Web Service Res 8:1–28. https://doi.org/10.4018/jwsr.2011040101
64. Decker G, Weske M (2007) Behavioral consistency for b2b process integration. In: International conference on advanced information systems engineering. Springer, Berlin, pp 81–95. https://doi.org/10.1007/978-3-540-72988-4_7
65. Zaha JM, Dumas M, Ter Hofstede A, Barros A, Decker G (2006) Service interaction modeling: bridging global and local views. In: 2006 10th IEEE international enterprise distributed object computing conference (EDOC'06), pp 45–55. https://doi.org/10.1109/EDOC.2006.50