Combining Goal-Oriented and BPMN Modelling to Support Distributed Microservice Compositions

Jesús Ortiz[a], Victoria Torres[b] and Pedro Valderas[c]
PROS Research Centre, Universitat Politècnica de València, Valencia, Spain

Keywords: Microservices, BPMN, Tropos, Goals, Transformation.

Abstract: Organizations usually use Business Processes (BPs) to describe how to achieve their goals. However, the decentralization found nowadays in many organizations force them to work with fragmented BPs that need to be coordinated to achieve these goals. In this context, microservices architectures are a good choice to coordinate such fragments. Nevertheless, these types of architectures increase the complexity of the underlying BPs since the control flow is split among the different microservices, and there is not a clear link among how each microservice participates in the achievement of each goal. In addition, one of the main challenges that developers face when creating a microservices composition is to identify the microservices that are required to support the organization's goals. To this end, in this paper, we propose to combine goal-oriented modelling with microservices compositions based on the choreography of BPMN fragments. The major contribution of this paper is the definition of a model-driven development approach to align both descriptions (goals and BPs) automatically through a model transformation that derives BPMN-based microservices compositions from goal diagrams. The main benefits of this solution are twofold: (1) to facilitate the distributed development of microservice compositions directed through goals, and (2) to help developers to maintain the composition aligned with the established goals when the composition evolves.

1 INTRODUCTION

Business processes (BPs) are the key instrument to organize and understand the interrelationships of the different activities in an organization to describe their goals (Weske, 2007). When these activities are performed in a decentralized way, e.g., by different departments within the same organization, microservices architectures turn into a very interesting and convenient way to implement such processes due mainly to their decoupling nature. Microservices architectures (Lewis, 2014) propose the decomposition of applications into small independent building blocks (the microservices) that focus on single business capabilities. Microservices can be deployed and maintained independently by different development teams, which leads to more agile developments and technological independence between them. When we want to support the goals defined in the BPs of organizations that use such architecture, microservices need to be composed. From the point of view of the software engineering field, two different approaches can be found in the traditional SOA architectures (Rosen, 2012) to coordinate the interactions between services: (1) *orchestration*, when the coordination is achieved from a single endpoint (Peltz, 2003), and (2) *choreography* when it is achieved in a decentralized way (Yahia, 2016).

Within microservices architectures, to keep a lower coupling and dependency among microservices for deployment and evolution, these compositions are usually implemented by means of event-based choreographies. The development team of each microservice is in charge of supporting the participation of the microservice in the event-based choreography in an independent and autonomous way. This solution, although improving the development independence demanded by this architecture, makes difficult to analyze the composition when maintenance or evolution is

[a] https://orcid.org/0000-0002-9352-1045
[b] https://orcid.org/0000-0002-2039-2174
[c] https://orcid.org/0000-0002-4156-0675

Ortiz, J., Torres, V. and Valderas, P.
Combining Goal-Oriented and BPMN Modelling to Support Distributed Microservice Compositions.
DOI: 10.5220/0012621000003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 75-86
ISBN: 978-989-758-696-5; ISSN: 2184-4895

required. This is because the control flow is split among different microservices, and there is not a clear link among how each microservice participates in the achievement of each organization's goal. In addition, one of the main challenges that developers face when creating a microservices composition is identifying the microservices that are required to support the organization's goals. The identification of microservices is a well-known problem in the research community because it is a complex, time-consuming, and error-prone task (Tizzei, 2017; Carvalho, 2020). Commonly, microservices architectures are derived from monolithic legacy systems, and developers must follow criteria such as cohesion, coupling, and communication between microservices to divide a monolithic system into microservices. Besides, identifying the microservices that are required to support the goals of an organization while defining the way they have to be composed makes this task even more complicated.

In this work, we present a model-driven development (MDD) approach that combines BPMN with goal-oriented modelling and achieves their synergy in order to improve these problems. This approach supports the creation of distributed microservices compositions based on event-based choreographies of BPMN fragments. On the one hand, goal-oriented modelling is used to help business process engineers better identifying the required microservices by analyzing the functional responsibilities that can be derived from the identified goals. To do so, we rely on the Tropos software development methodology (Castro, 2002) since it allows us to represent organization's goals with a high level of abstraction and also offers an easy-to-understand visual representation for developers who have little experience with goal modelling (Bresciani, 2002).

On the other hand, BPMN (Miers, 2008) is used to represent the microservices composition that is required to achieve the identified goals (Dietz, 2004). We use BPMN since it provides an intuitive and easy way to represent the semantics of complex processes and it is used by experts on the notation to define these processes, but also by other process stakeholders such as customers, marketing professionals, or finance employees that just need to analyze them (Nysetvold, 2006; Harmon, 2011; Andrade, 2016). The BPMN model is defined in two steps: first, a model transformation is applied to the Tropos diagram in order to obtain a preliminary BPMN model. This model represents the microservices composition in a global way, identifying the main functional responsibilities of microservices and the coordination required among them, but without defining the specific tasks that each microservice must perform. In the second step, this BPMN model is complemented by independent BPMN fragments that are created by the development team of each microservice. These BPMN fragments describe the tasks that each microservice must perform to fit its functional responsibilities. These BPMN fragments are executed through an event-based choreography, which provides the high level of independence and decoupling among microservices required by this type of architecture.

Thus, the main contributions of the proposed MDD approach are twofold:

- It facilitates the identification of the microservices that participate in a composition and helps developers relate the goals defined in a Tropos diagram to a BPMN process. This maintains the composition aligned with the established goals, which is a valuable mechanism to analyze the composition when requirements change, and the composition needs to evolve.
- It supports the distributed definition of a microservices composition through a set of independent BPMN fragments that must be created by the development team of each microservice. This provides a high level of autonomy and independence among development teams to create the whole composition collaboratively.

Note that the combination of BPMN with goal-oriented modelling has already been discussed by the scientific community (Alves, 2013; Horita, 2014; Koliadis, 2006). However, these works focused their efforts on orchestrated processes that are supported by monolithic systems. In our work, we focus on choreographed processes that are supported by systems deployed in distributed environments. Note also that the proposed approach supports very early stages of system development, where the specific system requirements are not yet clear. Therefore, the system domain is represented at a high level of abstraction. We focus on specifying the objectives to be achieved, without specifying details of how to achieve them, to generate an executable BPMN diagram that will be aligned with the defined goals.

The rest of this paper is organized as follows: Section 2 presents the proposed model-driven development approach to create distributed microservices compositions and explains the different steps that conform it. Section 3 exemplifies the benefits of our approach when a microservices composition based on the choreography of BPMN fragments needs

to be evolved. Section 4 analyses the related work. Finally, conclusions are commented on in Section 5.

2 MDD OF MICROSERVICES COMPOSITION

In this section, we present a MDD approach to create distributed microservices compositions by using a Tropos diagram and BPMN models. To this end, the following main three steps are proposed (see Figure 1):

1. **Tropos diagram construction.** In this step, the Business Engineer identifies the goals of the process of an organization and builds a Tropos diagram to represent them (section 2.1).
2. **BPMN collaboration diagram template creation.** In this step, a model transformation is automatically applied to derive a BPMN collaboration diagram template from the previously created Tropos diagram. The resulting BPMN model represents a choreography of BPMN fragments (section 2.2).
3. **BPMN fragment definition.** In this step, each microservice developer must complete its corresponding BPMN fragment to define the tasks that the microservice must perform to achieve its goals (section 2.3).

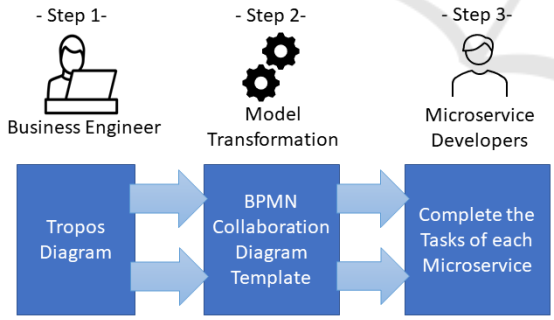

Figure 1: MDD approach for Microservices Composition.

To explain the main concepts of our approach, we use a running example based on the e-commerce domain. In this example, the system must manage the process of placing an order in an online shop, following the next sequence of actions: first, the system must register the client. If the client data is valid, the system checks the availability of the ordered items. If all the items are available, the system books the requested items and processes the payment with the client. Once the payment process has been successfully completed, the system updates the stock of the purchased items, creates a shipment order, and assigns it to a delivery company. Afterwards, the system updates the client record and informs the client about the shipment details. Then, the process finishes.

2.1 Definition of BP Goals with Tropos

To represent the goals of a business process we use Tropos (Bresciani, 2004), which is a goal-driven and agent-oriented language that aims to identify the motivations of software systems and the role that they will play in an organization (Hammer, 1994).

Models in Tropos are acquired as instances of a conceptual metamodel resting on several concepts. However, in this paper, we only focus on the ones used to integrate Tropos with our microservices composition approach, which are the following:

- ***Actor:*** It represents an entity that has strategic goals and intentionality within the system or the organizational setting.
- ***Goal:*** It represents actors' strategic interests. There are two types of goals: (1) *hard goals*, which are goals with a clear definition or criteria for deciding whether they are satisfied or not; and (2) *soft goals*, which are typically used to represent non-functional requirements. In this paper, we just focus on *hard goals* and therefore, the term goal is used to refer to a *hard goal.* The incorporation of *soft goals* to the approach presented in this paper is left as further work.
- ***Dependency:*** It indicates that one goal depends on another to be achievable.

To build a Tropos diagram, the first step is to identify the different actors that participate in a system process. To do so, we analyze the process to identify its core business functionalities. We propose to employ a data-driven strategy, dividing the process between the different data chunks that are managed during the process. For example, in the motivating example, we can identify the following data chunks: (1) the client data, (2) the stock data, (3) the payment data, and (4) the shipment data. Therefore, four actors can be defined as responsible of each data chunk: i.e., *Client Manager*, *Warehouse*, *Payment Manager* and *Distributor*.

Once the actors have been identified, we must relate them to goals. Goals can be defined as what the actor must achieve. Therefore, a goal is an abstraction of the actor's behavior. Figure 2 shows the Tropos diagram describing the *actors* (circles) participating in the running example, and their respective strategic high-level *goals* (round-corner rectangles linked to

them). We can identify the following goals for each actor: for the *Client Manager* actor, validate the customer and keep the customer data up to date; for the *Warehouse* actor, book the products and update the stock; for the *Payment Manager* actor, process the payment, and finally; for the *Distributor* actor, send the products to the client.

In addition to defining actors and goals, we can also identify that some goals depend on the completion of others in order to be achievable. For example, the *Book Products* goal of the *Warehouse* actor cannot be achieved until the customer is validated. Therefore, there is a dependency between the goal of the *Warehouse* actor and the goal of the *Client Manager*. A *dependency* between two goals is depicted by a solid arrow connecting them, where the goal that is pointed by the arrow is considered the *depender* goal (i.e., the goal that must be achieved first) and the goal at the other end is the *dependee* (i.e., the goal that depends on the completion of the previous one). This also allows designers to specify an order between the different goals.

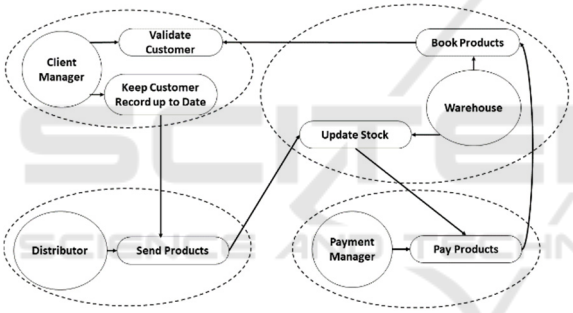

Figure 2: Representation example of a Tropos diagram.

Inspired by works such as (Greenwood, 2009), we propose to extend the goal representation in Tropos by defining a pre-condition and a post-condition for each goal. These conditions are based on the data required by a goal to be realizable (pre-condition) and achieved (post-condition). Therefore, the pre-condition will be related to the availability of data with a specific structure before the system tries to achieve a goal and the post-condition with the creation of data with a specific structure after a goal is achieved. Note that we consider data as a list of attributes defined as pairs key-value.

Table 1 represents the goals represented in Figure 2 with the associated pre- and post-conditions. For example, to achieve the *Validate Customer* goal, the system must receive the customer's data. In the same way, to consider this goal achieved, the system must know the status of the client (i.e., whether the client has been correctly validated or not).

Table 1: Goal definition.

Goal Name	Pre-condition	Post-condition
Validate Customer	**Name:** Customer Data **Data:** - Customer Name - Customer Address	**Name:** Customer Checked **Data:** - Customer Status (Valid | Not Valid)
Book Products	**Name:** Ordered Products **Data:** - Customer Status == Valid - Purchased Products (Name, Quantity)	**Name:** Stock Checked **Data:** - Products Status - Total Price
Pay Products	**Name:** Payment Data **Data:** - Products Status - Total Price - Payment Method	**Name:** Payment Checked **Data:** - Payment Status (OK | Fail)
Updated Stock	**Name:** Valid Payment **Data:** - Payment Status == OK	**Name:** Stock Updated **Data:** - Stock Status (OK | Fail)
Send Products	**Name:** Shipping Information **Data:** - Stock Status == OK - Customer Address	**Name:** Shipment Managed **Data:** - Delivery Company - Estimated Delivery Time
Keep Customer Record up to Date	**Name:** Products Shipped **Data:** - Delivery Company - Estimated Delivery Time	**Name:** Purchase Processed **Data:** - Products Status - Estimated Delivery Time

2.2 From Tropos to BPMN

Once the Tropos diagram has been defined, we can derive a structured BPMN collaboration diagram from it. This structured BPMN collaboration diagram represents a microservices composition. We use BPMN collaboration diagrams instead of BPMN choreography diagrams since collaboration diagrams allow us to separate the microservices that participate in a composition by business responsibilities and also, allow us to represent the dependencies between the different microservices. In addition, collaboration diagrams can be used to describe the internal behavior of each microservice together with the collaborative behavior of the whole composition. On the contrary,

in BPMN choreography diagrams it is more complex to separate microservices by business responsibilities, since they focus more on defining the composition from a global perspective, as well as the messages exchanged between the different participants. Furthermore, collaboration diagrams can be executed by most BPMN engines on the market while choreography diagrams are not supported (Corradini, 2018).

We have defined a model transformation to automatically generate a microservices composition template defined in a BPMN collaboration diagram from Tropos diagrams (see Algorithm 1). To achieve this, we need to consider that each actor in the Tropos diagram represents an entity that is in charge of a specific business responsibility (e.g., managing customer data, managing payment, and so on). By definition, a microservice is a building block that focuses on a specific business capability. Thus, we can consider that each actor in Tropos can be supported by a microservice. Considering that microservices are independent and autonomous components of a global system, we have decided to transform each actor of the Tropos diagram (and then each microservice) into a BPMN pool (line 2 of Algorithm 1). Figure 3 represents graphically how the transformation algorithm is applied to the running example. As we can see, the *Client Manager* actor is derived into the *Client Manager* BPMN pool, which represents a microservice. In the same way, the *Warehouse* actor is derived into the *Warehouse* pool.

To represent a goal in a BPMN pool, we use collapsed BPMN sub-processes. A collapsed BPMN sub-process is a group of tasks that performs a part of the entire process. In this case, we associate each goal with a collapsed BPMN sub-process to indicate that this goal can be achieved through the group of tasks represented by the sub-process. Therefore, each goal is derived into a collapsed BPMN sub-process (line 3). In Figure 3, the *Validate Customer* and *Book Products* goals are transformed into a collapsed BPMN sub-process with the same name.

The next step of the transformation is surrounding each sub-process with a catching intermediate message event as a previous element, and a throwing intermediate message event as a subsequent element (lines 4 - 6). In BPMN, these elements are used to indicate that a process either needs the reception of some message (catching) or is able to produce it (throwing). We use them to represent both the dependency between goals and the pre- and post-conditions of a goal in the BPMN model.

On the one hand, each dependency between two goals (line 9) is represented by connecting the throwing event after the sub-process that represents the *depender* goal with the catching event defined before the sub-process that represents the *dependee* goal. For instance, in Figure 3, the throwing event after the *Validate Customer* sub-process (*depender* goal) is connected to the catching event before the *Book Products* sub-process (*dependee* goal). After this step, if a catching intermediate message event of a pool is not connected to any throwing event, it is transformed into a catching start message event (line 11). In the same way, if a throwing intermediate message event of a pool is not connected to any catching event, it is transformed into a throwing end message event (line 12). For instance, the sub-process that represents the goal *Validate Customer* (without dependencies) is linked with a start catching message event.

On the other hand, note that goals have pre- and post-conditions that are associated with the availability and creation of specific data. To represent this in BPMN, the catching and throwing events that surround each sub-process are linked to a BPMN data object that defines the data required in the pre-condition and post-condition of the corresponding goal (lines 13 – 15). In Figure 3, the *Customer Data* data object is connected to the catching event of *Client Manager* to represent the pre-condition of the goal *Validate Customer* and the *Customer Checked* data object is linked to the throwing event to represent the post-condition of the same goal.

Finally, for each goal dependency (represented by two connected throwing and catching events), it is analyzed whether or not the post-condition of the *depender* goal (i.e., the data object associated with the throwing event of the corresponding sub-process) creates all the data required by the pre-condition of the *dependee* goal (defined in the data object associated to the catching event of the corresponding sub-process). In case the pre-condition of a *dependee* goal is not satisfied by the post-condition of a *depender* goal, the BPMN data objects associated with the catching and throwing events of the sub-process of the *depender* goal are extended with the data required by the *dependee* goal (lines 17 - 22). With this action, we are indicating the sub-process that represents the *depender* goal receives and propagates some data that is created earlier in the process in order to be used by a subsequent sub-process. For example, the *Book Products* goal (*dependee*) has a pre-condition that needs the list of the purchased products and the customer status. However, the post-condition of the *Validate Customer* goal (*depender*) only creates the customer status (see Table 1). Thus, the purchased products (propagated data) are added to the data objects of the catching and throwing events of the *Validate Customer* sub-process.

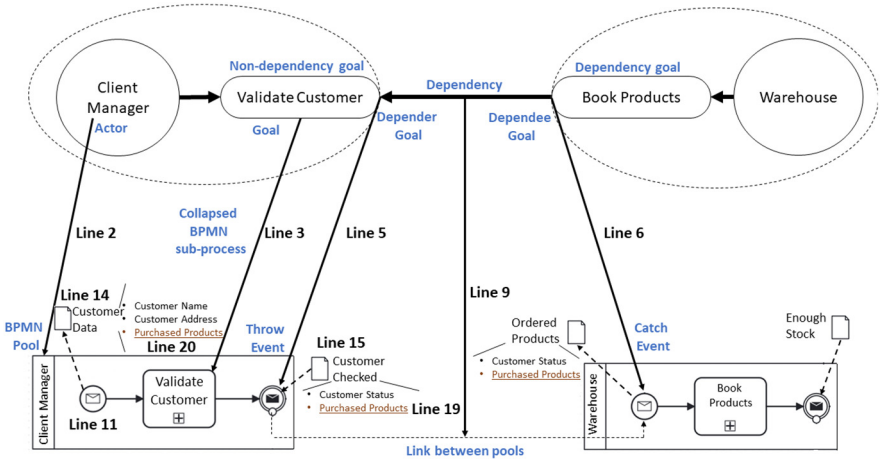

Figure 3: Established mappings between Tropos and BPMN.

INPUT: A Tropos diagram.
OUTPUT: A BPMN collaboration diagram.

1 **For each** actor in the Tropos diagram:
2 A BPMN pool is created;
3 Each goal defined is represented as a collapsed BPMN sub-process;
4 **For each** goal:
5 A throwing intermediate message event element is added as a succeeding element of the collapsed sub-process that represents the goal;
6 A catching intermediate message event element is added as a preceding element of the collapsed sub-process that represents the goal;
7 **End For**
8 **For each** goal with dependency:
9 The throwing intermediate message event of the *depender* and the catching intermediate message event of the *dependee* goal are linked to represent the interaction between goals;
10 **End For**
11 The catching intermediate message events that are not connected to any throwing intermediate message event, are transformed into a catching start message event;
12 The throwing intermediate message events that are not connected to any catching intermediate message event, are transformed into a throwing end message event;
13 **For each** catch/throwing event:
14 A BPMN data object is linked to the catching event to represent the pre-condition of its corresponding goal;
15 A BPMN data object is linked to the throwing event to represent the post-condition of its corresponding goal;
16 **End For**
17 **For each** catching event:
18 **If** the catching event receives less data than its throwing event sends:
19 The data is added to the throwing event which sends the data to the catching event;
20 The data is added to the initial catching event of the fragment that contains the throwing event;
21 **End if**
22 **End For**
23 **End For**

Algorithm 1: From Tropos to BPMN collaboration diagram.

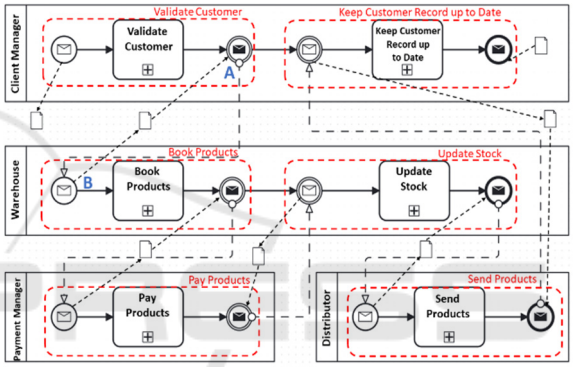

Figure 4: Obtained structured BPMN collaboration.

Figure 4 shows the resulting BPMN collaboration diagram when applying the transformation for the running example. It represents a microservices composition composed of four microservices that correspond to the four actors defined in the Tropos diagram: *Client Manager*, *Warehouse*, *Payment Manager,* and *Distributor*. Each pool includes as many collapsed BPMN sub-processes as goals linked to the actor. For example, the *Client Manager* microservice includes two collapsed BPMN sub-processes that correspond to the goals: *Validate Customer* and *Keep Customer Record up to Date*. Likewise, dependencies between goals have been supported by sending/receiving events between pools. For example, note how the dependency between the *Book Products* goal and the *Validate Customer* goal is supported by the throwing event of the *Client Manager* (see A in Figure 4) microservice and the catching event of the *Warehouse* microservice (see B in Figure 4).

Note that each pool must be executed by an autonomous microservice. Thus, the developers of

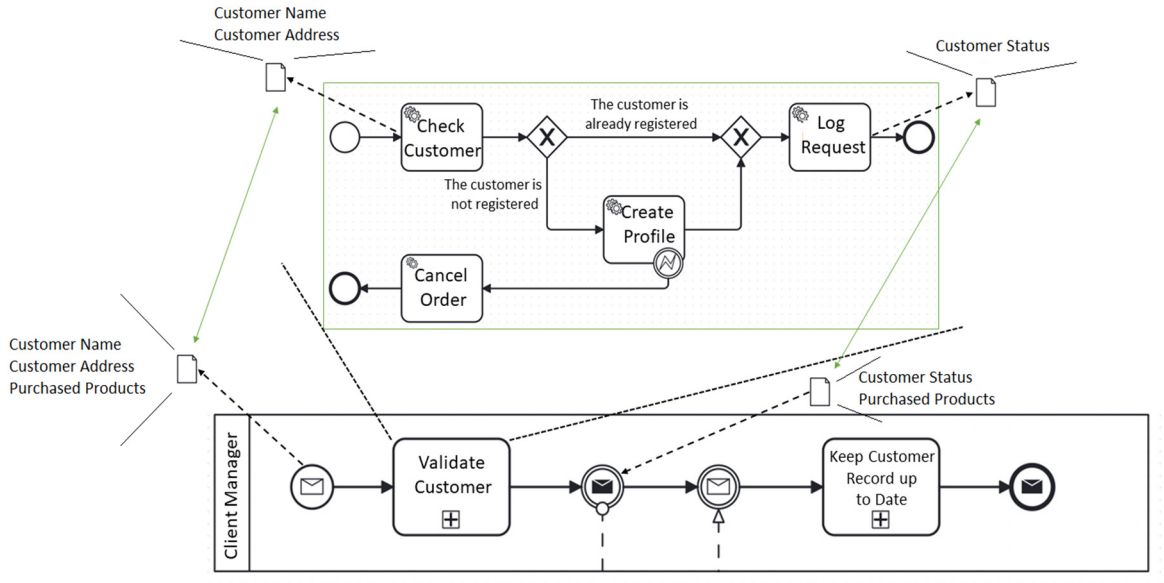

Figure 5: Refinement of the collapsed BPMN sub-process Validate Customer.

each microservice must complete them as we explain in the next sub-section.

2.3 Definition of Microservice Tasks

When the structured BPMN collaboration diagram has been generated, the next step is to specify the tasks that the collapsed BPMN sub-processes must perform to achieve the defined goals. Note that BPMN pools represent microservices, which are autonomous and independent software elements that can be developed by different development teams. Thus, the developers of a specific microservice can focus on the specification of the tasks of the sub-processes of the corresponding BPMN pool. Each BPMN sub-process can be developed independently of the others as long as they achieve the goals which are related, specifically as long as they achieve the pre-condition and post-condition of the goal.

For example, for the collapsed BPMN sub-process *Validate Customer* of the *Client Manager* microservice (see Figure 4), developers of this microservice may decide to define the following tasks (see Figure 5): *Check Customer* to begin the process of registration. If the customer is not registered, the *Create Profile* task is executed to register the new customer. If the customer completes the registration process or is already registered, the *Log Request* task stores the purchase made by the client and the sub-process terminates. An exception path is also added to cancel the purchase order if the customer does not want to perform the registration process with an exception boundary event and the *Cancel Order* task.

In the example represented in Figure 5, the *Validate Customer* BPMN sub-process is aligned with its related goal, i.e., the BPMN sub-process can reach the pre-condition and achieve the post-condition for the *Validate Customer* goal defined in Table 1. In the example, the *Client Manager* microservice receives an event that includes the customer's name, the customer's address, and the purchased products. Therefore, the BPMN sub-process of the example is aligned with the goal pre-condition. The customer's name and the customer's address are used by the tasks of the sub-process to check if the customer already exists and create a customer profile otherwise. In addition, at the end of the process, the customer status is created. Thus, the BPMN sub-process of the example is also aligned with the goal post-condition, as it creates all the data specified in the condition. Note also that the purchased products received by this sub-process are not used. This data is received at the beginning of the composition (when the *Client Manager* microservice receives the initial event) and must be propagated to the next sub-processes.

3 SUPPORTING EVOLUTION

The proposed model-driven approach allows developers to insert a group of tasks inside collapsed BPMN sub-processes. With our approach, microservices can be developed autonomously. Developers can define the microservice tasks

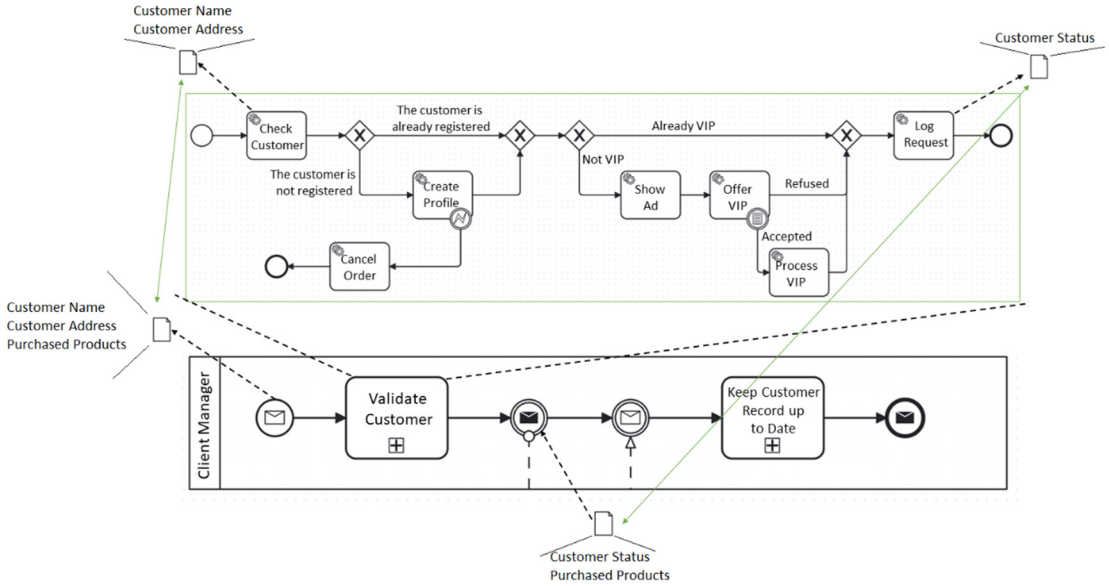

Figure 6: Modified Validate Customer collapsed BPMN sub-process.

independently. This approach offers developers another benefit: once a microservice has been developed, the development team of one microservice can change the tasks contained in a BPMN sub-process independently of the other participants, as long as the sub-process continues aligned with its established goals, i.e., developers can change the tasks contained in the collapsed BPMN sub-processes, but they cannot make modifications to the throw/catching events that surround the collapsed sub-processes, since they specify the pre- and post-conditions that the sub-process must achieve. Therefore, the only condition that developers must consider is that the evolved sub-process must still be able to achieve the pre-condition and the post-condition of the goal that it supports. Consequently, developers can perform changes in their BPMN fragments without the need to involve or coordinate them with other development teams that are developing other microservices. Note that microservices compositions are generally built to avoid dependencies and be reusable as much as possible. Our solution focuses on reuse within the domain of the system being designed. Therefore, the developed fragments can be reused by other systems that have domains equivalent to the designed system.

For example, if the *Client Manager* development team wants to differentiate between VIP clients and regular clients (if the store offers a premium service), the *Validate Customer* collapsed BPMN sub-process can be modified independently as follows (see Figure 6).

In this example, a new path is added to the *Validate Customer* sub-process to identify if the client is already VIP or not. In the case that is not a VIP client, two new tasks are added: *Show Ad* and *Offer VIP*, to offer the advantages of the premium service. Finally, the client can refuse or accept the offer. If it is accepted, a new task *Process VIP* is added to register the client in the premium program. This is considered a BPMN sub-process, that can be exchanged with the BPMN sub-process shown in Figure 5 at any time since the new sub-process is still aligned with the pre-condition and post-condition of its related goal (see Table 1). According to the pre-condition of the *Validate Customer* goal, the *Client Manager* microservice catches an event to receive the customer's data, the customer's address, and the purchased products. In the definition of the new sub-process, this data continues to be used by the *Check Customer* task, and therefore the pre-condition continues to be met. On the other hand, according to the post-condition of the *Validate Customer* goal, when the sub-process finishes all its tasks, the microservice must send an event that includes the customer status and the purchased products. This is also supported in the new sub-process since the *Log Request* task generates as a result of its process the customer status, and the purchased products are received by the initial catching Event of the *Client Manager* microservice. Consequently, the new sub-process is also aligned with the goal post-condition.

Therefore, the BPMN sub-processes can be modified independently from the rest of the

composition since they are developed from the local perspective of the microservice developer. In addition, since developers must continue to meet the pre-condition and post-condition of the goal, we ensure that the microservices composition remains aligned with the established goals if it evolves.

4 PROOF OF CONCEPT VALIDATION

In order to validate the proposed model-driven approach, we have developed the representative example and deployed the resulting microservices composition in a microservices architecture. The main goal of this preliminary validation was to evaluate whether the microservices compositions implemented by following the proposed steps can be executed correctly. Currently, the approach presented is implemented and integrated in a development environment that supports the different steps of the approach. Thus, we have validated the proposed model-driven approach as follows:

1. We created the Tropos diagram by using the CGM-Tool[1].
2. Once the goal model was created, we applied the proposed model transformation in order to obtain the BPMN model with the general view of the microservices composition.
3. Then, the authors of the paper played the role of microservice developers in order to independently create the BPMN sub-process that supports the goals represented in the previous BPMN model. To do so, the BPMN.io[2] modeler was used.
4. The microservices composition was deployed and executed in the architectural solution presented below.
5. We evaluated the correct execution by analyzing the logs generated by each microservice.
6. We evolved the microservices composition as explained in Section 3 and deployed it again to evaluate the execution of the new version.

Model Transformation Implementation. To perform the transformation from a Tropos diagram to a BPMN collaboration diagram, we have implemented a model transformation based on Algorithm 1 using Java[3]. We have used the CGM-Tool to generate the Tropos diagram, which represents the diagram in XML. Currently, there are several solutions to implement model transformations (Czarnecki, 2003). In this work, we have used a direct manipulation approach based on two parsers, one parser to read the XML generated by the CGM-Tool and the Java BPMN parser provided by Camunda.

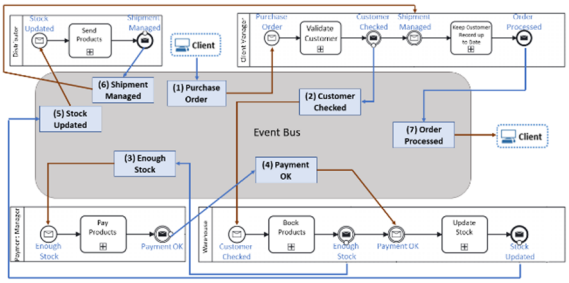

Figure 7: From BPMN collaboration diagram to BPMN fragments.

The Architectural Solution. [4] Once the microservices composition was completed it was deployed in a microservices architecture implemented as follows (Ortiz, 2022): the Spring Boot Java framework was used to implement all the microservices. Each microservice was endowed with a Camunda BPMN engine that oversees the execution of its respective BPMN fragment to execute (1) the sub-processes defined in its BPMN pool, and (2) the catch/throwing events to either receive or publish asynchronous events in a communication bus to support the collaboration with the rest of participants. This communication bus was supported by a RabbitMQ message broker.

Therefore, each microservice executes its corresponding BPMN fragment and informs other participants about its progress through the publication of events. In this way, the microservices composition is executed by means of an event-based choreography of BPMN fragments in which microservices wait for specific events to execute their corresponding piece of work (see Figure 7). Note that these events are named manually and allow data exchange between microservices. Following the motivation example, the resulting choreography begins when the composition receives the *Process Order* event. Then, each microservice performs its defined tasks and publishes its progress through events on the communication bus. The whole process ends when the *Client Manager* microservice has updated the client's data and sends him a notification to inform

[1] http://www.cgm-tool.eu/index.html
[2] https://bpmn.io/
[3] https://github.com/MicroservicesResearch/Tropos2BPMN

[4] Specific tool support to create this architectural solution is available at: https://github.com/microserviceresearch/microservices-composition-infrastructure

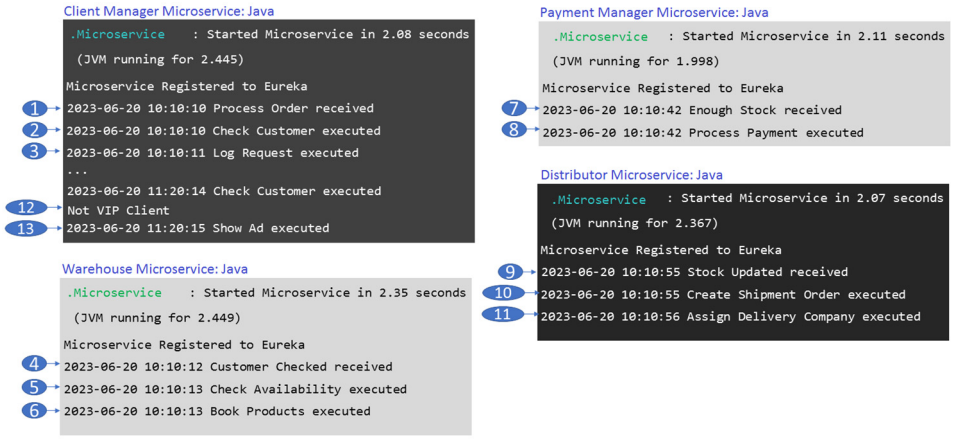

Figure 8: Logs generated in the composition deployment.

him that the purchase process has finished successfully through the *Order Processed* event.

Logs Evaluation. To validate the correctness of the executed microservices composition, we analyzed the logs generated by each microservice. In general terms, both the initial deployment of the microservices composition and the evolutions performed worked adequately. The evaluation consisted of checking if the choreography shown in Figure 7 was deployed correctly and if the microservices could correctly execute their processes and achieved their pre- and post-conditions (they received/sent the corresponding events). As a representative example, Figure 8 presents the logs generated in the deployment of the choreography shown in Figure 7. This figure illustrates the logs for each deployed microservice. Lines 1 through 3 show the correct execution of the *Client Manager* microservice, where it first receives the *Process Order* event and then executes the tasks defined in the *Validate Customer* sub-process (i.e., *Check Customer* and *Log Request*). In the same way, we can also observe the correct execution of the *Warehouse* (lines 4 through 6), *Payment Manager* (lines 7 and 8) and *Distributor* (lines 9 through 11) microservices. Additionally, the *Validate Customer* sub-process of the *Client Manager* microservice was modified as shown in Figure 6. The *Client Manager* microservice was re-deployed, executing the task *Show Ad* since the client was not VIP (lines 12 and 13). Therefore, we concluded that the evolution was executed correctly.

It is worth remarking that, as commented above, this constitutes a preliminary validation of the approach. However, a more precise evaluation is planned as further work.

5 RELATED WORK

In the research community, we can find several works that relate BPs with goals. We have classified these works in two different groups according to how this relation is achieved. The first group relates to the works that propose methods to relate goal diagrams with BPs. (Alves, 2013) proposes a model driven approach to obtain BPMN models from i* models. It proposes a heuristic process for mapping i* models to BPMN models but the execution order of the BPMN tasks obtained from the i* model must be manually defined by developers. In our work, the execution order of the tasks is automatically derived by the transformation algorithm. (Koliadis, 2006) proposes the GoalBPM methodology for relating business models to high level stakeholders' goals modelled using KAOS. In their work, to relate a BPMN model with a KAOS model, it is first necessary to create both models and then apply their proposed methodology to relate them. In our work, we derive a BPMN model from a goal diagram and at the same time we relate the goals with the processes. (Horita, 2014) proposes a transformation approach to transform KAOS models into BPMN models by using refinement patterns. The limitation of their work is that the KAOS models are defined at a low level, considering each goal directly a BPMN task. Consequently, the KAOS models can be considered as direct representations of BPMN models and vice-versa. In our work, we use collapsed BPMN sub-processes so that developers can define processes that can achieve the defined goals. In addition, it is not clear if their proposal can support more complex situation such as interaction between different actors. (Sabatucci, 2019) proposes an automatic approach that focuses on extracting goals from BPs but does not

address the reverse process, i.e., how to relate goals to BPs as we do in our work. In our work we focus on first defining the goal diagram, so that from the beginning the BPs are directed by the defined goals. (Brown, 2006) and (Kazhamiakin, 2004) present an approach to specify the requirements that a system must fulfil based on goal diagrams. These two approaches focus on modelling monolithic systems, and do not support distributed environments. Our proposal is focused on supporting distributed systems, i.e., an event-based microservices composition based on the choreography of BPMN fragments. (Huber, 2016) proposes to integrate semantic queries into process activities to support runtime discovery and dynamic invocation of goal-based IoT-services. This work also does not support distributed systems, and uses proprietary tools, while in our approach, since we use the BPMN standard, it can be integrated with different commercial tools that run BPMN models.

The second group relate to the works that propose to extend BPMN to explicitly define goals in the process models. (Braubach, 2010; Jander, 2011) propose an approach based on the notion of process goals to relate business goals to workflows. However, these approaches consider that the goals are embedded with the workflows and thus, they are not considered two different models, which limits their autonomy. A change in a BP must be directly translated to a change in the goals. These two works differ from ours in that we keep using the standard BPMN notation, which means that our BPMN descriptions can be executed in any BPMN engine (i.e., we are not tied to any proprietary tool). Furthermore, in our work, a modification in a BP does not have to directly affect to the goal diagram, if the modified BP still satisfies the pre- and post-conditions of its goal. (Greenwood, 2009) proposes an extension of the BPMN language to define goals in BPMN processes and also considers that the goals are embedded to workflows. This work also differs from ours in that their work introduces modifications to the BPMN language. In our work, we do not introduce any kind of complexity to the BPMN models, which were originally designed to describe processes and no other aspects as goals in this case. In fact, introducing new concepts into a well-known and consolidated notation can be risky since it can introduce complexity to the model (Zugal, 2011).

6 CONCLUSIONS AND FURTHER WORK

This paper explores how to combine goal-oriented modelling with microservices compositions based on the choreography of BPMN fragments, and achieve their synergy, benefiting from the advantages of both. For modelling business goals, we rely on the Tropos software development methodology. The contribution of this work is a model-driven development approach to develop distributed microservices composition directed through goals. For this purpose, we propose a model transformation to obtain a structured BPMN collaboration diagram from a Tropos diagram. The resulted BPMN collaboration diagram is composed through independent pools (the microservices), which can be developed by different development teams. Each microservice is made up of a set of collapsed BPMN sub-processes that developers must complete to define the tasks that the microservice must perform to achieve its related goals. The collapsed BPMN sub-processes can be changed without involving other microservices as long as the new sub-process continues fulfilling the pre-condition and the post-condition of its related goal, in order to maintain the composition aligned with the established goals.

As a future work, we want to develop the reverse process, i.e., to derive a Tropos diagram from a BPMN collaboration diagram that represents a microservices composition based on the choreography of BPMN fragments. In this way, the intrinsic goals of an existing microservices composition can be obtained. In addition, with the reverse process we can support more complex evolution scenarios to ensure that the composition, as it evolves, remains aligned with the established goals, i.e., allowing changes that not only affect the tasks that the microservices perform but also affect the communication between microservices (e.g., changes in throw/catching elements). In addition, we want to extend the current approach to also support *soft goals*, which we consider that can be very interesting in a distributed environment and extend our approach to consider cases such as legacy systems.

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 and funded with the Aid to First Research Projects (PAID-06-22), Research Vice-Rectorate of the Polytechnic University of Valencia (UPV).

REFERENCES

Alves, R., Silva, C., and Castro, J. (2013). *A bi-directional mapping between i* and BPMN models in the context of business process management.* ER@BR.

Andrade, E., van der Aa, H., Leopold, H., Alter, S., and Reijers, H. (2016). *Factors leading to business process noncompliance and its positive and negative effects: Empirical insights from a case study.*

Braubach, L., Pokahr, A., Jander, K., Lamersdorf, W., and Burmeister, B. (2010) *Go4flex: Goal-oriented process modelling.* In Intelligent Distributed Computing IV (IDC), Morocco (pp. 77-87). Springer Berlin Heidelberg.

Bresciani, P., and Sannicolò, F. (2002). *Applying Tropos Requirements Analysis for defining a Tropos tool.* In Agent-Oriented Information System (AOIS). In Fourth International Bi-Conference Workshop (pp. 135-138).

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). *Tropos: An agent-oriented software development methodology.* In Autonomous Agents and Multi-Agent Systems, 8, (pp. 203-236).

Brown, G., Cheng, B. H., Goldsby, H., and Zhang, J. (2006). *Goal-oriented specification of adaptation requirements engineering in adaptive systems.* In Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems (pp. 23-29).

Carvalho, L., Garcia, A., Colanzi, T. E., Assunção, W. K., Pereira, J. A., Fonseca, B., ... and Lucena, C. (2020). *On the performance and adoption of search-based microservice identification with tomicroservices.* In IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 569-580).

Castro, J., Klop, M., and Mylopoulos, J. (2002). *Towards requirements-driven information system engineering: the Tropos project.* Information Systems, vol. 27 (pp. 365-389).

Corradini, F., Morichetta, A., Polini, A., Re, B., and Tiezzi, F. (2018). *Collaboration vs. choreography conformance in BPMN 2.0: From theory to practice.* In IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC), (pp. 95-104).

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).

Dietz, J.L.G. (2004). *Basic Notions Regarding Business Processes and Supporting Information*. In Systems Proceedings of BPMDS'04, CAiSE'04 Workshops Proceedings, Riga, Vol. 2 (pp. 160-168).

Greenwood, D., and Ghizzioli, R. (2009). *Goal-oriented autonomic business process modelling and execution.* INTECH Open Access Publisher.

Hammer, M. and Champy, J. (1994). *Reengineering the Corporation – A manifesto for Business Revolution*. Nicholas Brealey Publishing.

Harmon, P., and Wolf, C. (2011). *Business process modeling survey.* Business process trends, 36(1), (pp. 1-36).

Horita, H., Honda, K., Sei, Y., Nakagawa, H., Tahara, Y., and Ohsuga, A. (2014). *Transformation approach from KAOS goal models to BPMN models using refinement patterns.* In Proceedings of the 29th Annual ACM Symposium on Applied Computing (pp. 1023-1024).

Huber, S., Seiger, R., Kühnert, A., Theodorou, V., and Schlegel, T (2016). *Goal-based semantic queries for dynamic processes in the internet of things.* International Journal of Semantic Computing, 10(02), (pp. 269-293).

Jander, K., Braubach, L., Pokahr, A., Lamersdorf, W., and Wack, K. J. (2011). *Goal-oriented processes with GPMN.* International Journal on Artificial Intelligence Tools, 20(06) (pp. 1021-1041).

Kazhamiakin, R., Pistore, M., and Roveri, M. (2004). *A framework for integrating business processes and business requirements.* In Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, (EDOC) (pp. 9-20).

Koliadis, G., and Ghose, A. (2006). *Relating business process models to goal-oriented requirements models in KAOS.* In Advances in Knowledge Acquisition and Management (PKAW) China, 7-8, Revised Selected Papers 9 (pp. 25-39). Springer Berlin Heidelberg.

Lewis, J., and Fowler, M. (2014). *Microservices*. https://martinfowler.com/articles/microservices.html (accessed December 2023).

Miers, D., and Stephen, W. (2008). *BPMN Modelling and Reference Guide*. Future Strategies Inc.

Nysetvold, A. G., and Krogstie, J. (2006). *Assessing business process modeling languages using a generic quality framework.* In Advanced Topics in Database Research, Volume 5 (pp. 79-93).

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

Peltz, C. (2003). *Web services orchestration and choreography.* Computer, 36(10), 46-52.

Rosen, M., Lublinsky, B., Smith, K. T., and Balcer, M. (2012). *Applied SOA: service-oriented architecture and design strategies.* John Wiley & Sons.

Sabatucci, L., and Cossentino, M. (2019). *Supporting dynamic workflows with automatic extraction of goals from BPMN.* ACM Transactions on Autonomous and Adaptive Systems (TAAS), 14(2), (pp. 1-38).

Tizzei, L. P., Nery, M., Segura, V. C., and Cerqueira, R. F. (2017). *Using microservices and software product line engineering to support reuse of evolving multi-tenant saas.* In Proceedings of the 21st International Systems and Software Product Line Conference-Volume A (pp. 205-214).

Weske, M. (2007). *Business Process Management: Concepts, Languages, Architecture.* Springer.

Yahia, E. B. H., Réveillère, L., Bromberg, Y. D., Chevalier, R., and Cadot, A. (2016). *Medley: An event-driven lightweight platform for service composition.* In Internat. Conf. on Web Engineering (pp. 3-20).

Zugal, S., Pinggera, J., and Weber, B. (2011). *Assessing process models with cognitive psychology.* Enterprise modelling and information systems architectures (EMISA).