Insights into the Grammar and Real User Connectivity for Choreography Patterns of Microservices-based Insurance Processes

Arne Koschel
Andreas Hausotter
Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hanover, Germany
Email: {arne.koschel, andreas.hausotter}
@hs-hannover.de

Christin Schulze
Alexander Link
Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hanover, Germany
Email: {christin.schulze, alexander.link}
@stud.hs-hannover.de

***Abstract*—To avoid the shortcomings of traditional monolithic applications, the Microservices Architecture (MSA) style plays an increasingly important role in providing business services. This is especially true for the insurance industry with its sophisticated cross-domain business processes. Here, the question arises of how workflows can be implemented to grant the required flexibility and agility and, on the other hand, exploit the MSA style's potential. There are two competing approaches to workflow realization, orchestration, and choreography, each with pros and cons. Though choreography seems to be the method of choice in MSA, it comes with some challenges. As the workflow is implicit – it evolves as a sequence of events being sent around – it becomes difficult to understand, change, or operate the workflow. To manage the challenges of the choreography approach, we use BPMN 2.0 choreography diagrams to model the exchange of domain events between microservices, which represent 'participants' in terms of BPMN. We aim to execute choreography diagrams automatically. For this, we developed a set of choreography patterns that represent frequently occurring sequences. We present the pattern language and discuss five patterns, a One-Way Task pattern, a Two-Way Task pattern, an Event-based Gateway – Deadline pattern, and an Open Parallel Gateway – different Senders pattern. Additionally, we present how a real user may get integrated into these interactions and provide insights into a grammar, which can validate the order of use for patterns in a given choreography. This paper is part of our ongoing research to design a microservices reference architecture for insurance companies.**

Keywords*—*Workflow; Choreography; BPMN; Patterns; Grammar; Business Processes; Microservice.

I. Introduction

In this article, which is an extended version of our earlier paper [1], we look at the realization of interactions within a microservices-based reference architecture for German insurance companies. Business workflows and multistep business processes are typical for insurance companies; see, for example, the reference architecture for German insurance companies (VAA) [2]. They are complemented by general regulations, such as the European GDPR [3], as well as insurance-specific laws and rules regarding, for example, financial regulations, data protection, and security [4].

Recently, the Microservices Architecture (MSA) style [5] [6] and cloud computing [7] became more and more interesting for insurance companies. Traditionally, several technologies from monolithic mainframe applications, functional decomposition-based software, traditional Service-Oriented Architectures (SOAs), which often utilize some kind of Enterprise Service Bus (ESB), Business Process and Workflow Management Systems (BPMS, WfMS) for orchestration, and 3rd party software, such as SAP software, were and are used together in insurance business applications, which implement their business processes.

Taking all those typical cornerstones from (over time grown) insurances into account, the goal of our currently ongoing research [8] is to develop a 'Microservice Reference Architecture for Insurance Companies (RaMicsV)' jointly with partner companies from the insurance domain. Within our work, we also look at the question: 'how to implement (insurance) business workflows with microservices, which potentially utilize several logical parts from RaMicsV'?

Within the MSA style, the more decoupled choreography is favored for this purpose [5] [6]. This is in some contrast, however, for example, to SOAs, where such workflows are mainly implemented using orchestration [9]. For example, one of our partner companies utilizes Camunda [10], another one a Java/Jakarta EE-based workflow tool.

However, since co-existence of all approaches is a 'must have' for our insurance partner companies, RaMicsV aims to address the *combined usage* of more traditional approaches and the MSA style, the combination of choreography and orchestration naturally comes to mind. As evolution is a key demand for our business partners – they can and will not just 'throw away' their existing application landscape – concepts such as orchestration and tools such as an ESB, whose use within MSA style architectures are both clearly disputable, have to be integrated reasonably well into our approach.

We thus started to look at the *combination* of choreography and orchestration, including a look at insurance domain specifics, in our work from [11]. In the present article, we will now have a focus on choreography-based approaches for (insurance) business processes. Particularly, we will examine an initial set of emerged *choreography patterns* for this pur-

pose, which we will model using choreography diagrams from the OMG BPMN 2.0 standard [12]. It should be noted that our goal is not a general implementation of choreographies, rather an implementation that orients itself toward real-world scenarios. Thus, we inspected multiple use cases from the insurance industry, one of which we will introduce later on.

In particular, we contribute in the present article our ongoing work and intermediate results about:

- The integration of the choreography within our RaMicsV;
- BPMN 2.0 choreography diagrams and the utilization of patterns;
- our pattern language for choreography patterns;
- four particular choreography patterns in depth, namely the One-Way Pattern, Two-Way pattern Open-Parallel Gateway – Different Senders pattern and the Event-based Gateway – Deadline pattern;
- User Connectivity within these interactions;
- insights into a grammar, to validate those patterns;
- and finally, insurance business use cases for those patterns.

The remainder of this article is structured as follows: After discussing related work in Section II, we briefly look at our current work within the RaMicsV context in Section III. Next, Section IV looks at BPMN 2.0 choreography diagrams with patterns. Section V then contributes our patterns usage and a pattern language for them, as well as four identified patterns, two of which are new to this version. Afterward, Section VI shows a possible way of integrating real users into these choreographies. Then, the current state of the choreography grammar will be presented in Section VII.

Moreover, Section VIII looks at a usage of those patterns within an insurance business use case. Finally, Section IX summarizes our results and concludes with some outlook to future work, with more patterns to follow.

II. Related Work

The basis of our research builds on authors in the scope of microservices, such as the work from Newman [6], as well as Fowler and Lewis [13]. Within the design of our reference architecture, we profit from different microservices patterns, as they are discussed by Krause [14] and Richardson [5].

To model our business processes, we use OMG's BPMN 2.0 specification. Also, we use as groundwork about business processes and its development with BPMN the works from Allweyer [15] [16], Rücker and Freund [17].

For the basics of service composition types, orchestration and choreography, we chose to rely on Decker's approach [18]. It is important that we define the choreography in terms of workflows within a microservices architecture. Quite many publications discuss the benefits of the choreography as a composition between (micro-)services. In particular, in several cases the theoretical benefit is presented or the combination of different approaches with the choreography is shown, as discussed by Rücker in his blog [19].

This paper ties in with our previous work on realizing a choreography [11]. In our last paper, we experimented with the implementation of a choreography using BPMN. The first pattern 'Any Problem becomes a Service' appeared to be difficult, since the monolithic BPMN does not support the message exchange between different microservices.

In Mikalkinas' [20] approach, a BPMN choreography diagram is transformed into a BPMN collaboration diagram and then executed. After this transformation, the BPMN collaboration diagram is executed by an engine, in this case Camunda [10]. We intend to bypass this conversion and provide direct execution of the choreography diagram. Thereby, our goal is to explore an implementation without an engine, since this corresponds to an orchestration in the case of Camunda.

Milanović and Gasević also try to implement choreography via BPMN and REWERSE II Rule Markup Language in their work [21]. They developed a rule-based extension for BPMN to realize choreography, called rBPMN. Ortiz et al. describe a similar approach [22]: In their work, rules are also defined on how to react based on which events in a choreography. This work uses fragments of BPMN. In both approaches (only) parts of the BPMN are considered, and in each case, only collaboration diagrams.

Another related approach is Richardson's SAGA pattern and the Eventuate Framework [5] [23]. The pattern describes the splitting of a transaction into several small local transactions. The local transactions trigger each other by messages/events. The error handling could become interesting for our further work. The framework includes two manifestations: Tram and Local. Eventuate Tram [24] so far only implements an orchestrated SAGA, so it does not yet include a choreography. Eventuate Local [25] provides event sourcing to store events. It also offers functions to perform transactions, through a publish/subscribe realization. It maps the technical implementation of a transaction rather than the communication and composition between services.

We try to implement a choreography in a more straight way as a compositional approach between microservices. Our vision is to use the choreography for the complete communication and workflow. We define the choreography as a global approach to processing a workflow without the intervention of a controlling part. This approach was described by us in our previous paper [11] and is also defined by Decker [18].

In order to evaluate interactions created from the choreography patterns using BPMN 2.0 choreography, we have created a grammar. For the creation of the grammar, we have used works by Vossen and Witt [26] and Priese and Erk [27] for advice.

To achieve this goal, we define patterns for BPMN choreography diagrams, which are supposed to be implemented automatically. To model our BPMN choreography diagrams, we used the framework chor-js developed by Ladleif et al. [28].

We do not focus on the processes within the (micro-) services themselves, rather only on the communication between them and the infrastructure. The use of patterns should also mitigate to some degree the complexity that can arise in (extensive) choreography-based workflows. The developed patterns borrow in structure and approach from Barros et al. [29].

III. Service-based Reference Architecture for Insurance Companies

This section presents our logical reference architecture for microservices in the insurance industry (RaMicsV) as initially started in [8].

RaMicsV defines the setting for the architecture and the design of a microservices-based application for our industry partners. The application's architecture will only be shown briefly, as it heavily depends on the specific functional requirements.

When designing RaMicsV, a wide range of restrictions and requirements given by the insurance company's IT management have to be considered. Regarding this contribution, the most relevant are:

- Enterprise Service Bus (ESB): The ESB as part of the SOA must not be questioned. It is part of a successfully operated SOA landscape, which seems suitable for our industry partners for several years to come. Thus, from their perspective, the Microservices Architecture (MSA) style is only suitable as an additional enhancement and only a partial replacement of parts from their SOA or other self-developed applications.
- Coexistence: Legacy applications, SOA, and microservices-based applications will be operated in parallel for an extended transition period. This means that RaMicsV must provide approaches for integrating applications from different architecture paradigms – looking at it from a high-level perspective, allowing an 'MSA style best-of-breed' approach at the enterprise architectural level as well.
- Business processes are critical elements in an insurance company's application landscape. To keep their competitive edge, the enterprise must change their processes in a flexible and agile manner. RaMicsV must therefore provide suitable solutions to implement workflows while ensuring the required flexibility and agility.

Figure 1 depicts the building blocks of RaMicsV which comprises layers, components, interfaces, and communication relationships. Components of the reference architecture are colored yellow; those out of scope are greyed out.

A component may be assigned to one of the following *responsibility areas*:

- **Presentation** includes components for connecting clients and external applications, such as SOA services.
- **Business Logic & Data** deals with the implementation of an insurance company's processes and their mapping to microservices, using various workflow approaches to achieve desired application-specific behavior.
- **Governance** consists of components that contribute to meeting the IT governance requirements of our industrial partners.
- **Integration** contains system components to integrate microservices-based applications into the industrial partner's application landscape.
- **Operations** consist of system components to realize unified monitoring and logging, which encloses all systems of the application landscape.
- **Security** consists of components to provide the goals of information security, i.e., confidentiality, integrity, availability, privacy, authenticity & trustworthiness, nonrepudiation, accountability, and auditability.

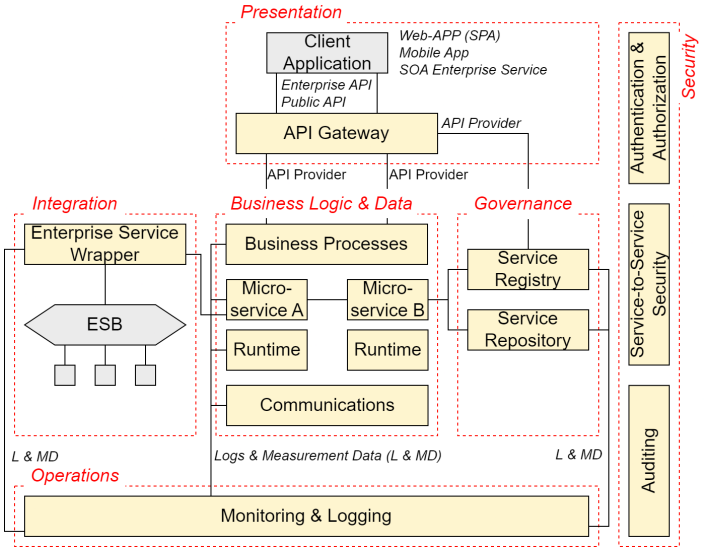

Figure 1. Building Blocks of the Logical Reference Architecture RaMicsV.

Components communicate either via HTTP(S) – using a RESTful API, or message-based – using a Message-Oriented Middleware (MOM) or the ESB. The ESB is part of the integration responsibility area, which itself contains a message broker (see Figure 1).

In the next section, we will take a look at the choreography in general and BPMN 2.0 choreography in particular as a lead-in to this paper's contribution, located in the responsibility area *Business Logic & Data*.

IV. Choreography

This section will present the core definition of choreography, as described in [11]. We briefly outline the use of BPMN, specifically the choice of BPMN 2.0 choreography diagrams.

A. Choreography

In a choreographed system, there exists no central coordinator, unlike in orchestration [30]. Decker [18] describes the definition of a choreography as a global view of how services

cooperate and the interaction between participants. This proves to be a challenge when modeling and monitoring a workflow, as the workflow is mapped by the interaction between the participants. It follows that the responsibility of executing and processing the workflow is transferred to each participant [31].

While choreography may be combined with other patterns, like the event-driven architecture [32], we decided not to focus on technical implementations yet, but will eventually.

B. BPMN 2.0 choreography

BPMN 2.0 choreography is chosen as the modeling language, since BPMN is also used by our partners. In the BPMN specification exist at least three significantly different diagram types to describe processes:

- **Process** known as classic BPMN. It visualizes the entire process.
- **Collaboration** splits a classic process into multiple participants (or microservices). Each sub-process in a participant can be recognized, but also the message exchange between the participants.
- **Choreography** which visualizes only the exchange of messages between participants.

In contrast to our previous work [11], we now focus only on the implementation of BPMN 2.0 choreography diagrams [12], as they visualize the interaction between microservices. In these diagrams, a participant represents a microservice. We aim to execute business processes using a choreographed MSA. Choreography serves as a global composition pattern [18]. We start with a collaboration diagram to map the whole process, which we then transform into a choreography diagram to focus on the communication. The processes within the participants are out of scope as we focus on the means of communication.

To automatically implement the choreography with BPMN 2.0 choreography, we develop patterns that map frequently occurring sequences. It should be a wide selection of things that must, should or can occur. The pattern language and the yet-to-be-developed grammar will be used to create a tool that automatically accepts modeled choreography diagrams and generates the necessary infrastructure and message exchange.

V. Choreography Patterns

In this section, we will present a pattern language, as well as two patterns from our list. The language intends patterns to be assembled to produce more extensive use cases. The patterns originate from real-world use cases.

A. Pattern Language

A pattern language is utilized to describe the patterns uniformly. It consists of the following elements (cf. [7]):

- **Identification number (ID)** of the pattern.
- **Name** of the pattern.
- **Figures** that visualize the pattern. Consisting of BPMN 2.0 choreography diagrams, BPMN collaboration diagrams, and UML Sequence diagrams.
- A **Description** which describes the use, content, and flow of the pattern.
- **Rules** and conditions under which the pattern may be used.
- A list of **used BPMN elements** from the choreography- and collaboration diagrams, as named in [12].
- **Used Patterns**, which this pattern builds upon.
- **Synonyms** and similar patterns from literature and industry.
- **Variations** where the core concept of the pattern stays the same.
- **Typical combinations** and patterns with high compatibility.
- Example **Use-Cases** from the industry.

B. One-Way Task

Now that the pattern language has been introduced, we start with the most atomic pattern, the *One-Way Task*.

- **ID:** BPMNChor01
- **Name:** One-Way Task
- **Figures:** See Figure 2, Figure 3, and Figure 4.
- **Description:** Participant A wants to deliver a message to Participant B. The initiator (A) sends the message to the receiver (B).
- **Rules:** None.
- **Used BPMN Elements:** startEvent (none), messageStartEvent, participant (pool), Message originating from the initiator, endEvent (none).
- **Used Patterns:** None, this pattern is atomic and depicts the minimum amount of interaction.
- **Synonyms:** Fire-and-Forget, One-Way Notification
- **Variations:** None.
- **Typical combinations:** Due to the atomic properties of this pattern, it may be combined with every other pattern.
- **Use-Case:** Sending an E-Mail or push-notification. For a longer scenario, see Section VIII.

C. Two-Way Task

The *Two-Way Task* describes a bidirectional message exchange between two participants within a task.

- **ID:** BPMNChor02
- **Name:** Two-Way Task
- **Figures:** See Figures 5, 6, and 7.
- **Description:** Participant A initiates a message exchange with participant B. Participant A waits for participant B's response.
- **Rules:** None.
- **Used BPMN Elements:** startEvent (none), messageStartEvent, participant (pool), Message originating from the initiator, endEvent (none).

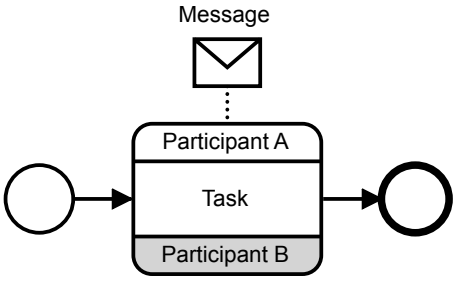

Figure 2. One-Way Task Choreography.

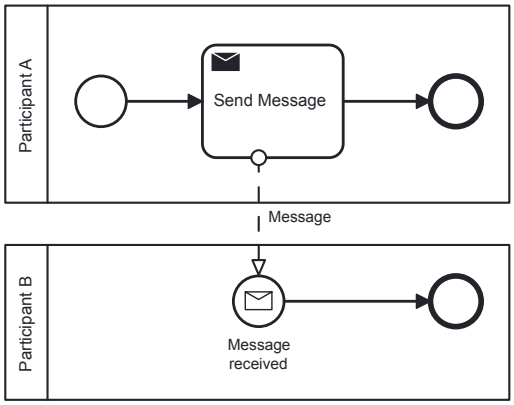

Figure 3. One-Way Task Collaboration.

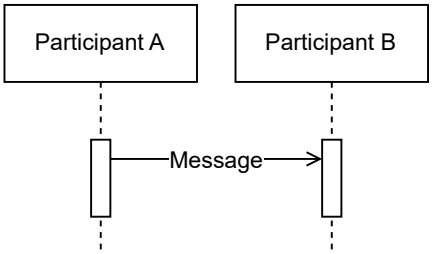

Figure 4. One-Way Task UML Sequence.

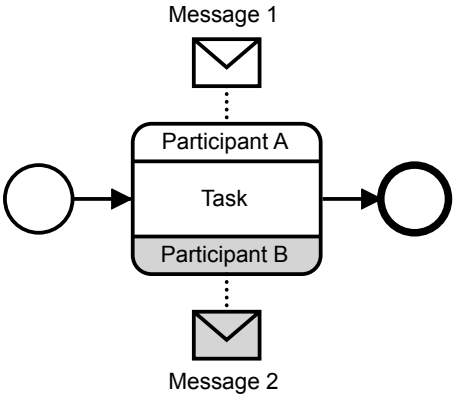

Figure 5. Two-Way Task Choreography.

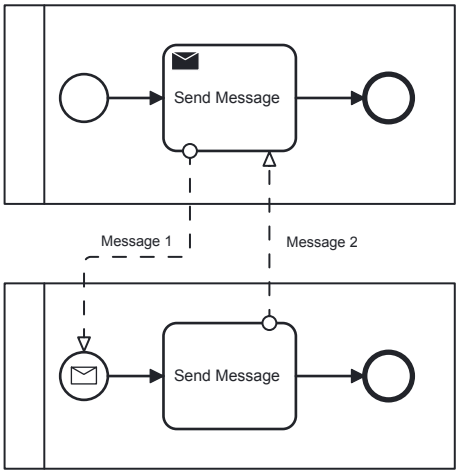

Figure 6. Two-Way Task Collaboration.

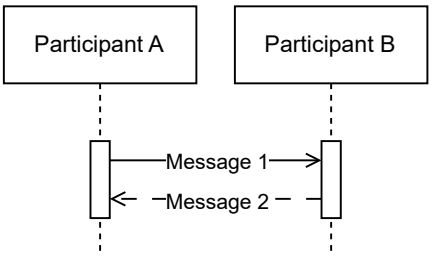

Figure 7. Two-Way Task UML Sequence.

- **Used Patterns:** Two-Way Task is an extension of the One-Way Task. The extension adds the response from participant B.
- **Synonyms:** synchronous request-response.
- **Variations:** None.
- **Typical combinations:** Due to the atomic properties of this pattern, it may be combined with every other pattern.
- **Use-Case:** Message exchange via SMS. For a longer scenario, see Section VIII

This concludes the One-Way Task and the Two-Way Task as the minimal way of communication, next we will introduce Event-based Gateway – Deadline pattern.

D. Event-based Gateway – Deadline

The *Event-based Gateway – Deadline* pattern describes a more complex, yet often occurring, scenario where the flow of a process is determined by a temporal aspect.

- **ID:** BPMNChor11
- **Name:** Event-based Gateway – Deadline
- **Figures:** See Figure 8, Figure 9, and Figure 10.
- **Description:** An answer only has a limited time frame to be received. Participant B receives a message from Participant A. Participant B has to answer within a given timeframe (N-Time) or else another workflow will be triggered. Participant A has the timing responsibility.
- **Rules:** Participant B has to initiate the answering message. A Two-Way communication is required.

- **Used BPMN Elements:** startEvent (none), messageStartEvent, participant (pool), Message, originating from the initiator, messageStartEvent, timerStartEvent, endEvent (none).
- **Used Patterns:** This pattern is based upon the *Sequence Flow – Two Participants* pattern (to be published) with the restriction that the receiving participant has to answer in the given timeframe.
- **Synonyms:** Asynchronous Request-Response
- **Variations:** None.
- **Typical combinations:** This pattern may be inserted into any request-response workflow when a timing-based component is needed.
- **Use-Case:** Setting a Deadline for paying an invoice. If the time is over, a reminder may be sent. For a longer scenario, see Section VIII.

E. Open Parallel Gateway – Different Senders

The *Open Parallel Gateway – Different Senders* divides the interaction into two paths, with different senders (initiators). The order of the tasks in the respective paths can take place in parallel, they have no fixed order.

- **ID:** BPMNChor09
- **Name:** Open-Parallel Gateway – Different Senders
- **Figures:** See Figures 11, 12, 13.
- **Description:** The parallel gateway with different senders visualizes a parallel message dispatch of the participants activated in the task before the gateway.
- **Rules:** Participant B has to initiate the answering message. A Two-Way communication is required.
- **Used BPMN Elements:** startEvent (none), messageStartEvent, participant (pool), Message, originating from the initiator, messageStartEvent, timerStartEvent, endEvent (none).
- **Used Patterns:** None.
- **Synonyms:** Fire-and-Forget, Publish/Subscribe
- **Variations:** There can also be responses from the receivers, since the gateway is not associated with any participant, it is never closed.
- **Typical combinations:** Basic patterns can be used within the gateway.
- **Use-Case:** Send payment request and send policy, see Section VIII.

VI. Real User Connectivity

Another challenge in choreography is to implement communication with a real user, such as a staff member or a customer. In an orchestrated environment (e.g., using Camunda [10]), the orchestrator would take over the management of communication and also the tasks for the respective user (called tasklist).

Since this coordinator does not exist in the choreography, we have decided to abstract the user via an agent service. Accordingly, another service is added to realize the communication with the user. We are currently planning to implement

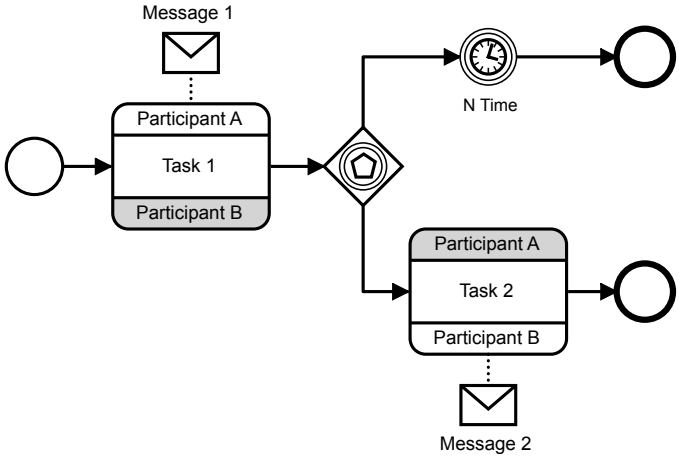

Figure 8. Event-based Gateway – Deadline Choreography.

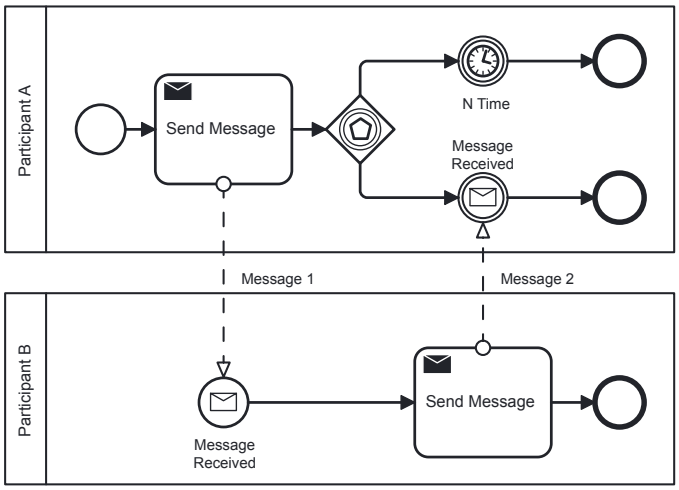

Figure 9. Event-based Gateway – Deadline Collaboration.

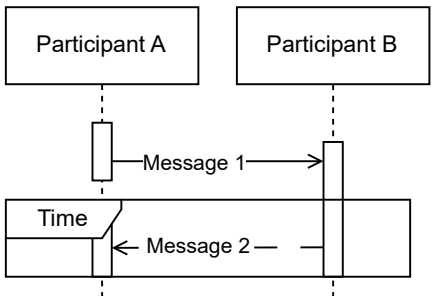

Figure 10. Event-based Gateway – Deadline UML Sequence.

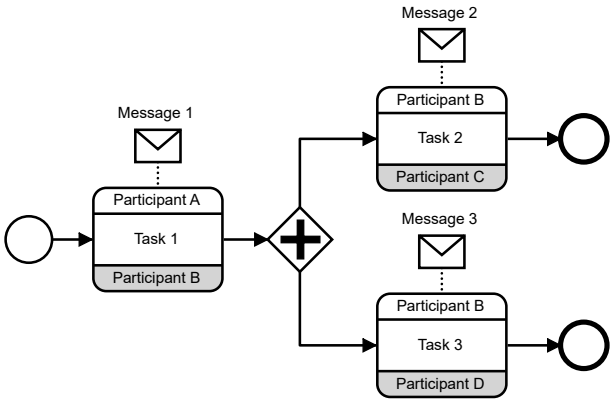

Figure 11. Open-Parallel Gateway – Different Senders Choreography.

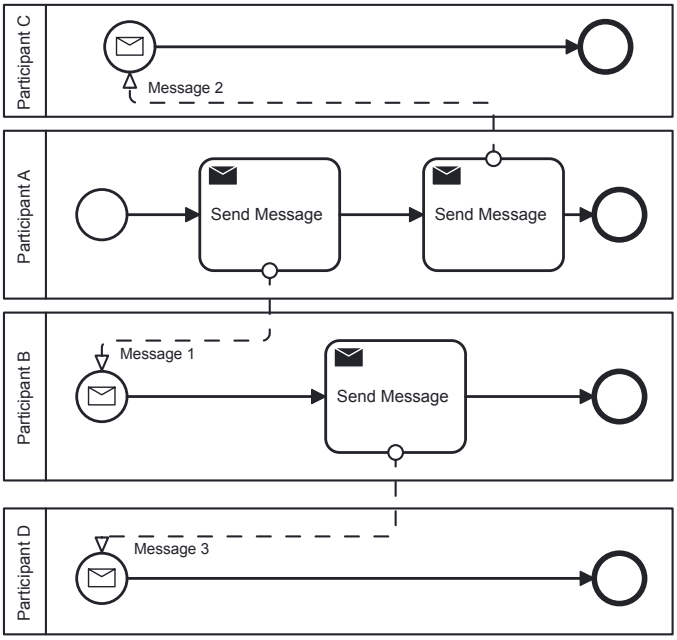

Figure 12. Open-Parallel Gateway – Different Senders Collaboration.

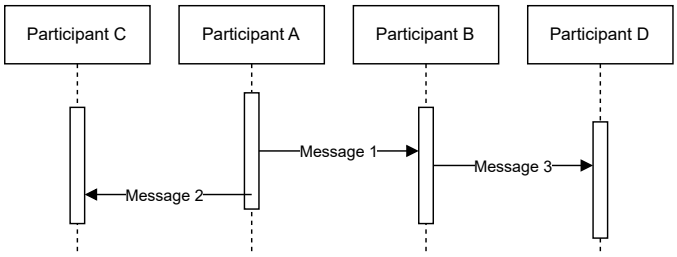

Figure 13. Open-Parallel Gateway – Different Senders UML Sequence.

this via a mail interface, but any other type of interface is conceivable in the future.

As an initial example, some information is sent to a customer (see Figure 14). The service represents an automated service within an MSA. In this case, the agent also simulates a task list for the user.

The actual implementation is visualized in Figure 15. In addition to the service and the customer, a customer agent is introduced that takes care of sending and receiving mail.

Sending mails is implemented in such a way that the agent service listens for events for the respective user, reacts to incoming events, generates and sends mails from these events. Receiving mails works the opposite way. As soon as a customer sends a mail to the agent service, the agent service converts the mail into an event and publishes it. Services within the MSA can then receive and process these events.

The agent service is not limited to an actual implementation type. In the first iteration, we decided to use a mail service to test the feasibility of this realization.

VII. Choreography Grammar

We are currently creating a grammar to specify the allowed interactions. This grammar validates the order of the patterns used to create valid interactions and prohibit invalid ones. However, some rules coming from the patterns, for example requirements for the participants, cannot be verified using the grammar without overcomplicating it. Therefore, additional constraints may be necessary.

The basic idea is that the letters of the grammar represent the patterns and BPMN elements so that a created word is an interaction to represent a choreography. Currently, it is a context-free language, since all relations R correspond to the form $N \times (N \cup T)^*$ [26]. The grammar contains all the patterns in the catalog.

Due to the complexity of the language and the rule set, only the non-terminals, terminals and the rules that describe the patterns from Section V are introduced here.

A. Notations $^+$ and *

The Kleene plus notation is used to extend the Kleene star notation for the grammar. This helps keep the derivation rules more concise [26].

- Using X^+ for at least once, or more
- Using X^* for arbitrary amount, or zero

B. ANY

ANY is a placeholder non-terminal for any other non-terminal of the pattern language, except start and end events. The use of *ANY* makes the grammar more concise.

It can be replaced by epsilon ε, i.e., an empty letter. Furthermore, there is no $ANY \Rightarrow ANY\ ANY$ within Rule 2, as precise rules are specified for each pattern as to when which patterns can come before or after.

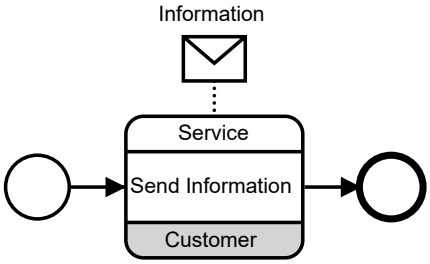

Figure 14. Abstract Example of Real User Connectivity — Choreography.

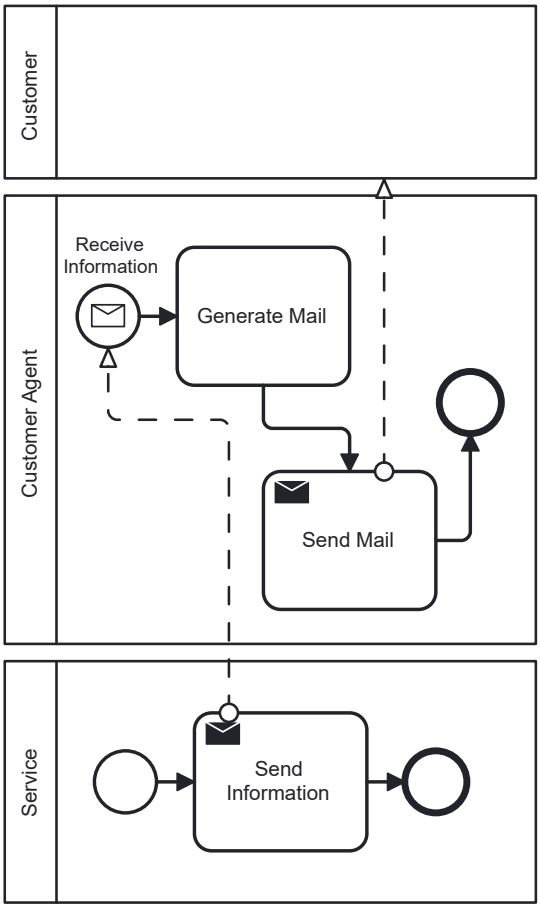

Figure 15. Abstract Example of Real User Connectivity — Collaboration.

C. Start and end event

Start and end are not necessary in a choreography. We force them into the grammar to make them more explicit.

- Non-Terminals: E_s for **s**tart event and E_e for **e**nd event
- Terminals: e_s for **s**tart event and e_e for **e**nd event
- Therefore, Rule 3: $E_s \Rightarrow e_s$, and Rule 4: $E_e \Rightarrow e_e$

D. Atomic patterns

The atomic patterns are described as follows:

- Described by One-Way Task and Two-Way Task.
- Non-Terminals: T_{ow} for **O**ne-**W**ay Task and T_{tw} for **T**wo-**W**ay Task
- Terminals: t_{ow} for **O**ne-Way Task and t_{tw} for **T**wo-Way Task
- Therefore, Rule 5: $T_{ow} \Rightarrow ANY\ t_{ow}\ ANY$ and Rule 6: $T_{tw} \Rightarrow ANY\ t_{tw}\ ANY$

E. Gateway patterns

To differentiate the gateways and also the type of open or closed, there are several non-terminals and terminals for creating a gateway and one universal terminal for closing, i.e., g_{close}. This terminal defines whether paths are merged again. As we have only used open gateways in our research work so far, we will not go into this implementation in more detail now, but in future work.

1) Event-based Gateway - Deadline: The Event-based Gateway — Deadline pattern has the following structure:

- Non-Terminal: G_{rd} for **r**ace-condition and **d**eadline
- Terminal: g_{rd}
- Using Rule 7: $G_{rd} \Rightarrow$ $ANY\ (t_{ow}|t_{tw})^+\ ((g_{rd}[ANY\ E_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))$
- where $ANY\ (t_{ow}|t_{tw})^+$ is an exchange starting the race condition,
- and $(g_{rd}[ANY\ E_e])$ as the path that can/should be executed within the defined time,
- and $(g_{rd}[e_{it}\ ANY\ E_e])$ as a path that describes the timer (i.e., e_{it}) and executes something and/or ends the process after expiry.

2) (Open-)Parallel Gateway: Since there is no verification of senders and receivers within the choreography task, there is no differentiation in the grammar between the Open-Parallel Gateway — Same Senders and Different Senders patterns. In addition, the implementation using the terminal g_{close} means that there is no direct distinction between open and closed gateways at the time they are created. The following describes the realization of an open parallel gateway:

- Non-Terminal: G_p
- Terminal: g_p
- Using Rule 1: $C \Rightarrow E_s\ ANY\ G_p$
- and using Rule 8: $G_p\ \Rightarrow$ $ANY\ (t_{ow}|t_{tw})^+\ (g_p[ANY\ E_e])^+$
- where $ANY\ (t_{ow}|t_{tw})^+$ as an exchange starting the parallel message flow,
- and $(g_p[ANY\ E_e])^+$ as the definition of the parallel paths.

F. Derivation Rules

The derivation rules for the grammar are defined as follows: $R = \{$

1) $C \Rightarrow E_s\ ANY\ (G_x|G_p)$,
2) $ANY \Rightarrow T_{ow}|T_{tw}|G_p|G_{rd}|\varepsilon$,
3) $E_s \Rightarrow e_s$,
4) $E_e \Rightarrow e_e$,
5) $T_{ow} \Rightarrow ANY\ t_{ow}\ ANY$,
6) $T_{tw} \Rightarrow ANY\ t_{tw}\ ANY$,

7) $G_{rd} \Rightarrow$
$ANY\ (t_{ow}|t_{tw})^+\ ((g_{rd}[ANY\ E_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))$,
8) $G_p\ \Rightarrow$
$ANY\ (t_{ow}|t_{tw})^+\ (g_p[ANY\ E_e])^+$,

}

G. Structure of each interaction

Each choreography follows the same structure: C is the start symbol of the choreography. The default structure is: $C \Rightarrow E_s\ ANY\ E_e$. Alternatively, the structure may be as follows: $C \Rightarrow E_s\ ANY\ (G_x|G_p)$ because in the case of open gateways (**ex**clusive or **p**arallel) there exists more than one end event.

Therefore, the choreography grammar (G) is currently described by $G = (N, T, R, S)$ with:

- $N = \{ANY, E_s, E_e, T_{ow}, T_{tw}, G_{rd}, G_p\}$ as Non-terminals, like a choreography, any patterns, or a specific pattern defined by a non-terminal shortcut,
- $T = \{e_s, e_e, e_{it}, t_{ow}, t_{tw}, g_{rd}, g_{gp}\}$ as terminals, for a specific pattern defined by a terminal shortcut,
- R as rules for the derivation from non-terminal to terminal, which are defined in Section VII-F,
- $S = C$, as a starting form of a choreography

The grammar is still in progress, so the content may change. After the presentation of the theoretical concepts, we will look at practical application in the next section.

VIII. Pattern Scenarios in Insurance Companies

To realize the pattern language of the two introduced patterns in Section V completely, this section evaluates use cases of the patterns from the insurance industry and shows the implementation of the user and grammar.

We consider a typical process where a new insurance application is managed. The process *New Insurance Application* adopted from Freund and Rücker [17], but can also be taken directly from the insurance business model of our partners in the insurance industry, thus mapping a real-world use case. Due to the size of the process, it is only briefly described below and the parts containing the patterns are further explained.

A. Process and patterns

In the process, a customer submits a new insurance application. If the request is rejected, this information is noted in the backend and the customer is informed. If the request is accepted, a policy is created. After creation, the policy is sent and the customer is requested to submit the first payment. If the payment is not made within 60 days, the request, and the policy are invalid. If the customer pays in time, the insurance is valid.

The process starts with a synchronous communication using *Two-Way Task* pattern. The Application Service and Policy Service are then informed about the status of the interaction and the process splits into two parallel (or two independent)

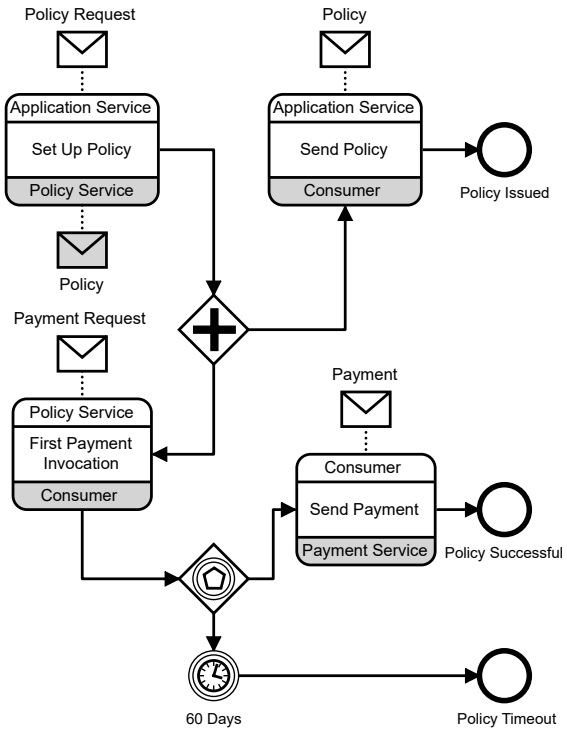

Figure 16. New Insurance Application Process – Cutout.

paths, implemented by the *Parallel Gateway — Different Senders* pattern.

In one path, the *One-Way Task* pattern is represented, by sending the policy to the customer. In the other path, the *Event-based Gateway – Deadline* pattern is utilized by the sending and receiving of the payment request.

The process excerpt describes the happy path (see Figure 16), i.e., the customer obtains an insurance policy. It starts with the creation of the policy by the Application Service submitting a request to the Policy Service. The Policy Service responds by sending the policy. This interaction is realized using *Two-Way Task* pattern.

The interaction is then divided into two independent paths using the *Parallel Gateway — Different Senders* pattern, which means that the sequence of tasks in the respective paths can take place in parallel, their order is not fixed.

In the right path, the *One-Way Task* pattern is implemented. The application service sends the policy to the client. After sending, the task is completed and the path ends.

The *Event-based Gateway – Deadline* pattern is shown in the bottom path. The policy service sends the first payment request to the client. Then a timer is started. If the customer pays within 60 days, the policy, and the process are successful. If the customer does not pay within 60 days, a timeout occurs and the policy becomes invalid.

As shown with the payment request and the incoming payment in Figure 16, the *Event-based Gateway — Deadline* pattern contains the *One-Way Task* pattern. It shows that this

fundamental pattern is the basis of the minimal communication for the choreography.

B. User Connectivity

This subprocess also involves communication with a user, in this case the consumer. The Policy Service, which is an automated service within the MSA, sends the consumer a payment request. The consumer has to respond to this payment request within 60 days by paying the first insurance premium. Otherwise, the insurance status will expire.

The corresponding task in the choreography diagram is within the *Event-based Gateway – Deadline* pattern in the lower path of the *Parallel Gateway — Different Senders* pattern. This example shows the two paths, communication to the outside world and from the outside world to the system. The conversion to the outside world is described in Figure 17. The payment service generates a message, which the agent service converts into an email and sends to the consumer.

The implementation from the outside world is described in Figure 18. The consumer pays the first fee and an email is sent to the system, which is received by the agent service. The agent service converts the mail into a message and sends it to the policy service.

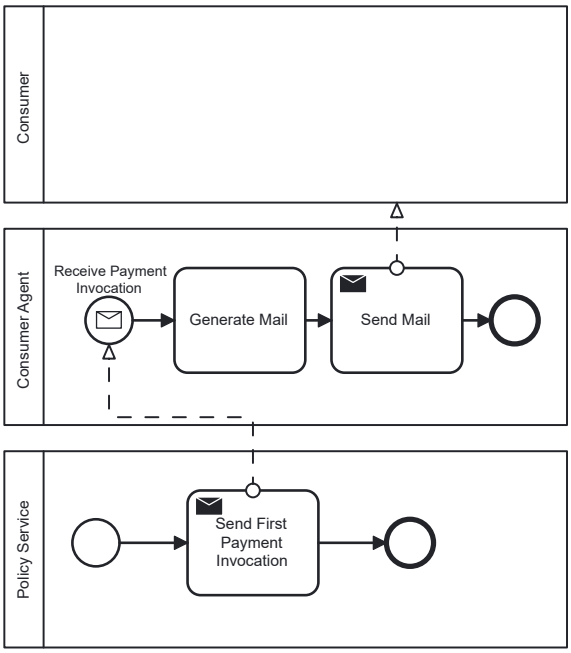

Figure 17. Realization User connection — to the outside world.

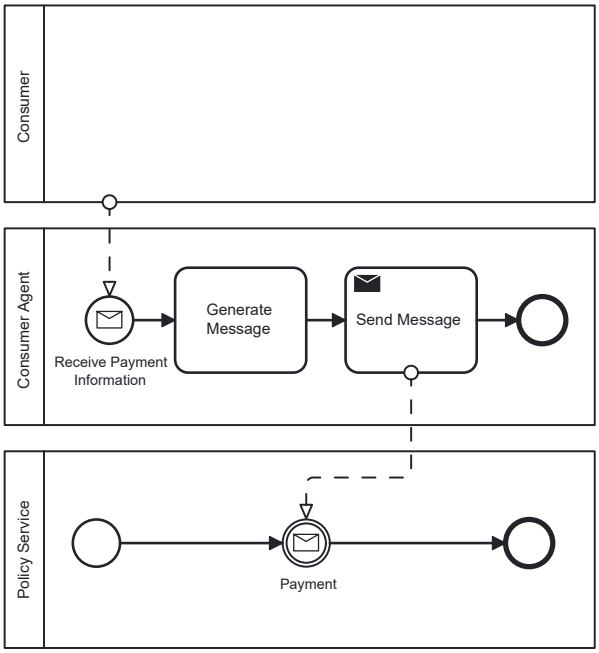

Figure 18. Realization of User connection - from the outside world.

C. Grammar

Section VII described the grammar limited to the patterns presented here. Now the implementation is presented using a scenario. As the subprocess is only an extract, we represent the flow into the task *Set Up Policy* as a start event.

1) Start with the Rule 1: $C \Rightarrow E_s\ ANY\ G_p$
2) Derive $E_s \Rightarrow e_s$ using Rule 3 and $ANY \Rightarrow \varepsilon$ using Rule 4, so that $C \Rightarrow e_s\ G_p$
3) Derive $G_p \Rightarrow\ ANY\ (t_{ow}|t_{tw})^+\ (g_p[ANY\ E_e])^+$ using Rule 8,
   so that $C \Rightarrow e_s\ ANY\ (t_{ow}|t_{tw})^+\ (g_p[ANY\ E_e])^+$
4) Select a Two-Way Task in front of the gateway and set ANY to ε using Rule 4 after the start, so that
   $C \Rightarrow e_s\ t_{tw}\ (g_p[ANY\ E_e])^+$
5) Create two paths, so that
   $C \Rightarrow e_s\ t_{tw}\ (g_p[ANY\ E_e])(g_p[ANY\ E_e])$
6) Replace ANY in the first path using Rule 3 with a One-Way Task (*Send Policy* Task) and $E_e \Rightarrow e_e$ using Rule 4, so that
   $C \Rightarrow e_s\ t_{tw}\ (g_p[T_{ow}\ e_e])(g_p[ANY\ E_e])$
7) Replace One-Way Task Non-Terminal with its Terminal using Rule 5, so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[ANY\ E_e])$
8) Replace ANY in the second path with an Event-based Gateway — Deadline (*Payment Procedure* Task) using Rule 2, so that
   $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[G_{rd}\ E_e])$
9) Derive G_{rd} using Rule 7, so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])\ (g_p[ANY\ (t_{ow}|t_{tw})^+\ ((g_{rd}[ANY\ E_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))])$
10) Replace ANY before the first path of the race condition with ε using Rule 2 and select One-Way Task (*First Payment Invocation* Task), so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[(t_{ow}\ ((g_{rd}[ANY\ E_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))])$
11) Replace ANY in the first path of the race condition with a One-Way Task (*Send Payment* Task) using Rule 2, so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[(t_{ow}\ ((g_{rd}[T_{tw}\ E_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))])$
12) Derive the first path of the race-condition to the ter-

minals using Rule 6 and Rule 4, so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[(t_{ow}\ ((g_{rd}[t_{tw}\ e_e])\ (g_{rd}[e_{it}\ ANY\ E_e]))])$

13) Derive ANY in the second path with ε using Rule 2 and replace the non-terminal from the end event using Rule 4 so that $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[(t_{ow}\ ((g_{rd}[t_{tw}\ e_e])\ (g_{rd}[e_{it}\ e_e]))])$

The expression $C \Rightarrow e_s\ t_{tw}\ (g_p[t_{ow}\ e_e])(g_p[(t_{ow}\ ((g_{rd}[t_{tw}\ e_e])\ (g_{rd}[e_{it}\ e_e]))])$ describes the process as shown in Figure 16.

IX. Conclusion and Future Work

The effective modeling and implementation of business processes is of crucial importance for an insurance company. Coming from BPMN notation, there needs to be a concise way of realizing the modeled process in the MSA style using the choreography. In this article, we further solidified the beginning of our choreography pattern language as the first steps towards a clear realization approach with precise implementation rules to map from BPMN diagrams to the distribution of microservices.

A way of integrating real users has been presented, to handle user interaction and allows the abstraction of these interactions. The grammar can validate the order of pattern usage, but cannot validate the rules of each individual pattern, an extra step for rule validation may be needed.

Several more patterns are needed to cover a broader range of different business use cases in the insurance industry. We also plan to evaluate all theoretical patterns with our insurance industry partners to ensure practical use. Additionally, an extension of the XML schema of BPMN diagrams is needed to serialize the patterns and validate them for correct usage. In future work, we will thus present additional patterns and an extension to the BPMN XML schema. We will also aim to refine our choreography pattern language and evaluate its additional benefit through a concrete implementation.

References

[1] C. Schulze, A. Link, H. Meyer, A. Koschel, and A. Hausotter, "Towards Patterns for Choreography of Microservices-based Insurance Processes," in *SERVICE COMPUTATION 2023, 15th Intl. Conf. on Advanced Service Computing*. IARIA, ThinkMind, 2023, pp. 1–7, Online. Available: https://www.thinkmind.org/index.php?view=article&articleid=service_computation_2023_1_10_10003 [retrieved: 02, 2024].

[2] Gesamtverband der Deutschen Versicherungswirtschaft e.V. - General Association o.t. German Insurance Industry, "VAA Final Edition. Das Fachliche Komponentenmodell (VAA Final Edition. The Functional Component Model)," 2001.

[3] European GDPR, "Complete guide to GDPR compliance," Online. Available: https://gdpr.eu/ [retrieved: 02, 2024].

[4] Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) - Federal Financial Supervisory (BaFin), "Versicherungsaufsichtliche Anforderungen an die IT (VAIT) (Insurance Supervisory Requirements for IT (VAIT)) vom 03.03.2023," 2023, Online. Available: https://www.bafin.de/DE/Startseite/startseite_node.html [retrieved: 02, 2024].

[5] C. Richardson, *Microservices Patterns: With examples in Java*. Shelter Island, New York: Manning Publications, 2018.

[6] S. Newman, *Building microservices: designing fine-grained systems*. Sebastopol, California: O'Reilly Media, Inc., 2015.

[7] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Vienna, 2014.

[8] A. Koschel, A. Hausotter, R. Buchta, A. Grunewald, M. Lange, and P. Niemann, "Towards a Microservice Reference Architecture for Insurance Companies," in *SERVICE COMPUTATION 2021, 13th Intl. Conf. on Advanced Service Computing*. IARIA, ThinkMind, 2021, pp. 5–9, Online. Available: https://www.thinkmind.org/articles/service_computation_2021_1_20_10002.pdf [retrieved: 02, 2024].

[9] A. Hausotter, A. Koschel, M. Zuch, J. Busch, and J. Seewald, "Components for a SOA with ESB, BPM, and BRM – Decision Framework and architectural Details," *Intl. Journal od Advances in Intelligent Systems*, vol. 9, no. 3 & 4, pp. 287–297, 2016.

[10] "Workflow and decision automation platform," Nov 2021, Online. Available: https://camunda.com/ [retrieved: 02, 2024].

[11] A. Koschel, A. Hausotter, R. Buchta, C. Schulze, P. Niemann, and C. Rust, "Towards the Implementation of Workflows in a Microservices Architecture for Insurance Companies – The Coexistence of Orchestration and Choreography," in *SERVICE COMPUTATION 2023, 14th Intl. Conf. on Advanced Service Computing*. IARIA, ThinkMind, 2023, pp. 1–5, Online. Available: https://www.thinkmind.org/index.php?view=article&articleid=service_computation_2023_1_10_10002 [retrieved: 02, 2024].

[12] OMG, *Business Process Model and Notation (BPMN), Version 2.0*, Object Management Group Std., Rev. 2.0, January 2011, Online. Available: http://www.omg.org/spec/BPMN/2.0 [retrieved: 02, 2024].

[13] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," 2014, Online. Available: https://martinfowler.com/articles/microservices.html [retrieved: 02, 2024].

[14] L. Krause, *Microservices: Patterns and Applications: Designing fine-grained services by applying patterns*. Lucas Krause, 2015.

[15] T. Allweyer, *Kollaborationen,Choreographien und Konversationen in BPMN 2.0 - Erweiterte Konzepte zur Modellierung übergreifender Geschäftsprozesse - Collaborations,Choreographies and Conversations in BPMN 2.0 - Advanced Concepts for Modeling Comprehensive Business Processes*. Fachhochschule Kaiserslautern, 2009.

[16] T. Allweyer, *Geschäftsprozessmanagement: Strategie, Entwurf, Implementierung, Controlling. - Business process management: strategy, design, implementation, controlling.* W3l GmbH, 2005.

[17] B. Rücker and J. Freund, *Praxishandbuch BPMN 2.0 - Practice Handbook BPMN 2.0*. Carl Hanser Verlag München Wien, 2014.

[18] G. Decker, O. Kopp, and A. Barros, *An Introduction to Service Choreographies*, vol. 50, no 2 ed. Information Technology, 2008.

[19] B. Rücker, "The Microservices Workflow Automation Cheat Sheet," 2018, Online. Available: https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa[retrieved: 02, 2024].

[20] D. Mikalkinas, *Situation-aware Modelling and Execution of Choreography*. Stuttgart University, 2015.

[21] M. Milanović and D. Gasević, "Modeling service choreographies with rule-enhanced business processes," in *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, 2010, pp. 194–203.

[22] J. Ortiz, V. Torres, and P. Valderas, "A catalogue of adaptation rules to support local changes in microservice compositions implemented as choreographies of bpmn fragments," 2023, Online. Available:https://riunet.upv.es/bitstream/handle/10251/181551/CatalogueOfAdaptationRules.pdf?sequence=1 [retrieved: 02, 2024].

[23] C. Richardson, "Eventuate Framework," 2021, Online. Available: https://eventuate.io/[retrieved: 02, 2024].

[24] ——, "Eventuate Tram," 2021, Online. Available: https://eventuate.io/abouteventuatetram.html[retrieved: 02, 2024].

[25] ——, "Eventuate Local," 2023, Online. Available: https://github.com/eventuate-local/eventuate-local[retrieved: 02, 2024].

[26] G. Vossen and K.-U. Witt, *Grundkurs Theoretische Informatik (Basic Theoretical Computer Science)*, 6th ed. Wiesbaden: Springer Vieweg, 2016, publication status: Published.

[27] L. Priese and K. Erk, *Theoreotische Informatik - Eine umfassende Einführung (Theoretical computer science - A comprehensive introduction)*, 4th ed. Berlin, Heidelberg: Springer Vieweg, 2018, publication status: Published.

[28] J. Ladleif, A. von Weltzien, and M. Weske, "chor-js: A modeling framework for bpmn 2.0 choreography diagrams," 2019.

[29] A. Barros, M. Dumas, and H. A.H.M, "Service interaction patterns," 2005, pp. 302–318, Online. Available:http://www.workflowpatterns.com/documentation/documents/serviceinteraction_BPM05.pdf [retrieved: 02, 2024].

[30] C. Chen, "Choreography vs orchestration," Online. Available: https://medium.com/ingeniouslysimple/choreography-vs-orchestration-a6f21cfaccae [retrieved: 02, 2024].

[31] B. Rücker, "The Microservices Workflow Automation Cheat Sheet," Online. Available: https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa [retrieved: 02, 2024].

[32] ——, *Practical Process Automation - Orchestration and Integration in Microservices and Cloud Native Architectures*. O'Reilly, 2021.