Towards Patterns for Choreography of Microservices-based Insurance Processes

Alexander Link
Henrik Meyer
Christin Schulze
Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hanover, Germany
Email: andreas.hausotter@hs-hannover.de

Andreas Hausotter
Arne Koschel

Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hanover, Germany
Email: arne.koschel@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 gets hard 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 two patterns, a One-Way Task pattern, and a Event-based Gateway – Deadline pattern. This paper is part of our ongoing research to design a microservices reference architecture for insurance companies.**

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

I. INTRODUCTION

Business workflows and multistep business processes are typical for insurance companies; see, for example, the reference architecture for German insurance companies (VAA) [1]. They are complemented by general regulations, such as the European GDPR [2], as well as insurance-specific laws and rules regarding, for example, financial regulations, data protection, and security [3].

Recently, the Microservices Architecture (MSA) style [4] [5] and cloud computing [6] 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 [7] 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 [4] [5]. This is in some contrast, however, for example, to SOAs, where such workflows are mainly implemented using orchestration [8]. For example, one of our partner companies utilizes Camunda [9], 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 [10]. 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 purpose, which we will model using choreography diagrams from the OMG BPMN 2.0 standard [11]. 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;

 ISBN: 978-1-68558-043-8

- BPMN 2.0 choreography diagrams and the utilization of patterns;
- our pattern language for choreography patterns;
- two particular choreography patterns in depth, namely the One-Way Pattern and the Event-based Gateway – Deadline pattern;
- 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 two identified patterns. Moreover, Section VI looks at a usage of those patterns within an insurance business use case. Finally, Section VII 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 [5], as well as Fowler and Lewis [12]. Within the design of our reference architecture, we profit from different microservices patterns, as they are discussed by Krause [13] and Richardson [4].

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 [14] [15], Rücker and Freund [16].

For the basics of service composition types, orchestration and choreography, we chose to rely on Decker's approach [17]. 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 [18].

This paper ties in with our previous work on realizing a choreography [10]. 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' [19] 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 [9]. 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 [20]. They developed a rule-based extension for BPMN to realize choreography, called rBPMN. Ortiz et al. describe a similar approach [21]: 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 [4] [22]. 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 [23] so far only implements an orchestrated SAGA, so it does not yet include a choreography. Eventuate Local [24] 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 [10] and is also defined by Decker [17].

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 from Ladleif et al. [25]. 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. [26].

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 [7].

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

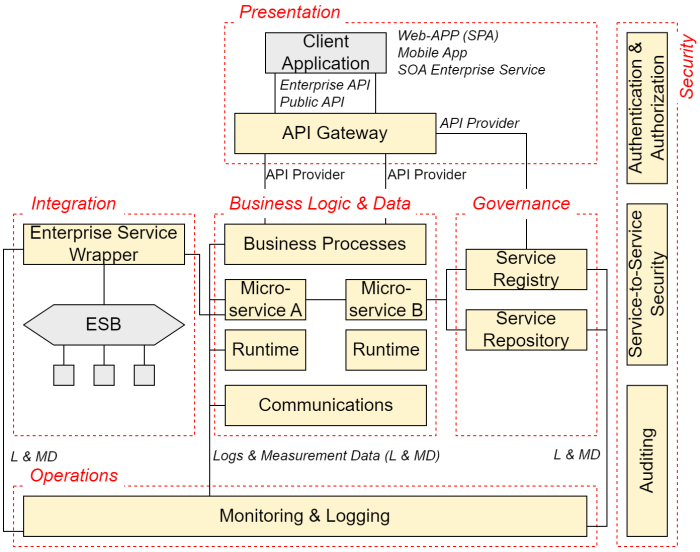

Figure 1. Building Blocks of the Logical Reference Architecture RaMicsV.

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.

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 have 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 [10]. 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 [27]. Decker [17] 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 [28].

While choreography may be combined with other patterns, like the event-driven architecture [29], 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.

 ISBN: 978-1-68558-043-8

In contrast to our previous work [10], we now focus only on the implementation of BPMN 2.0 choreography diagrams [11], 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 [17]. 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 realize 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. [6]):

- **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 [11].
- **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.

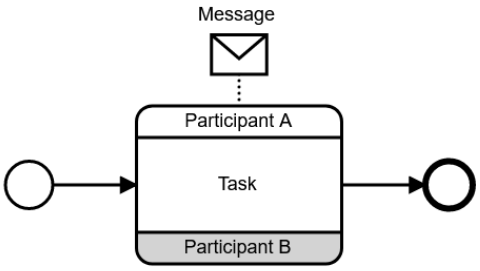

Figure 2. One-Way Task Choreography.

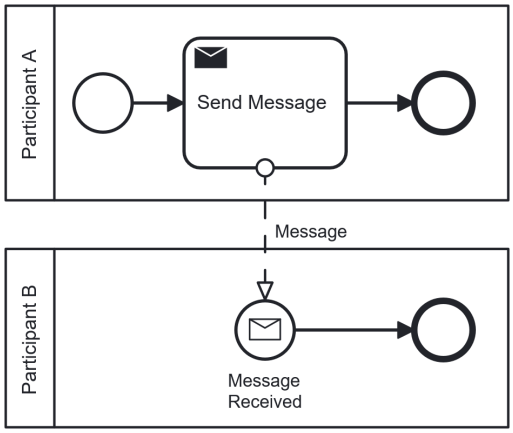

Figure 3. One-Way Task Collaboration.

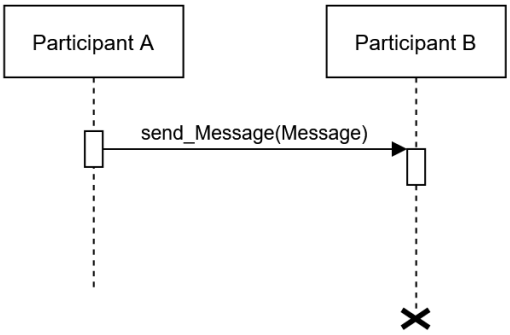

Figure 4. One-Way Task UML Sequence.

- **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.

 ISBN: 978-1-68558-043-8

- **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 VI

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

C. 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 5, Figure 6, and Figure 7.
- **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 VI.

VI. 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.

We consider a typical process where a new insurance application is managed. The process *New Insurance Application* adopted from Freund and Rücker [16], 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.

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

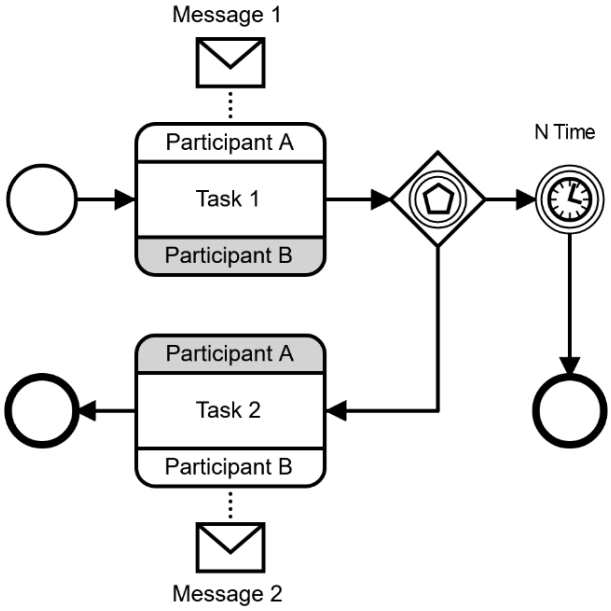

Figure 5. Event-based Gateway – Deadline Choreography.

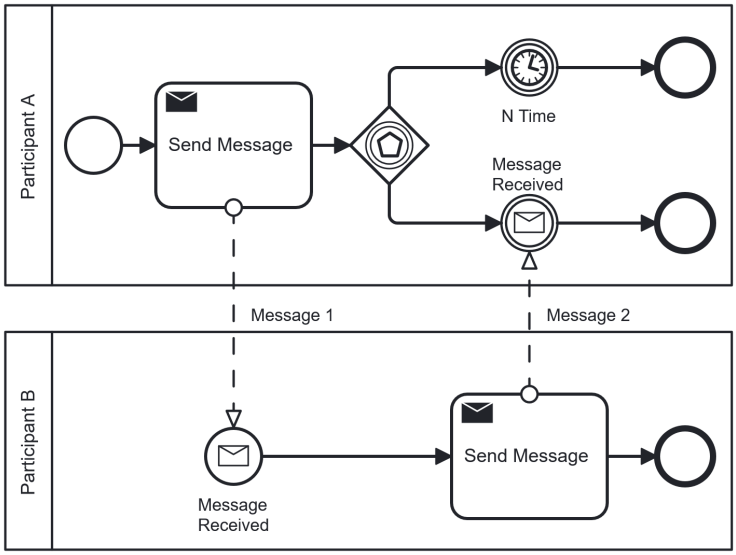

Figure 6. Event-based Gateway – Deadline Collaboration.

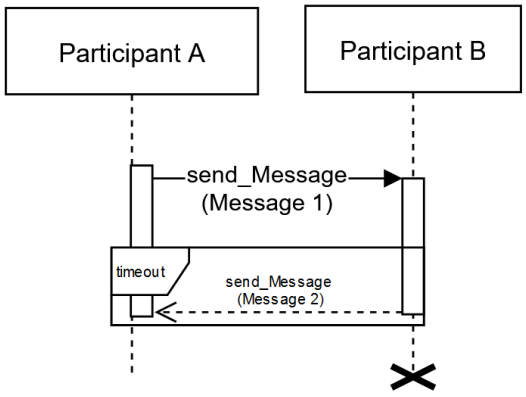

Figure 7. Event-based Gateway – Deadline UML Sequence.

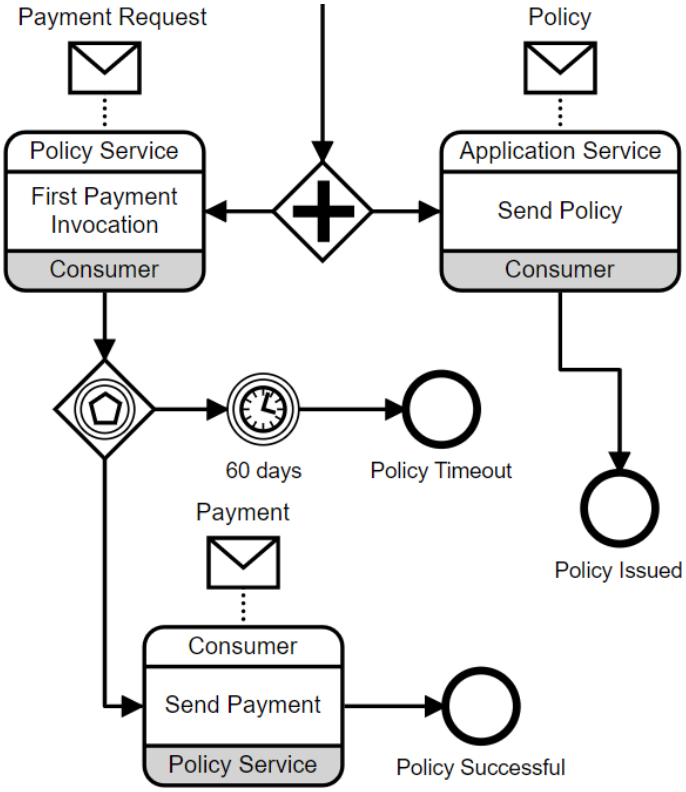

Figure 8. New Insurance Application Process – Cutout.

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 parallel flow represents the examples of the use cases in the insurance industry. In this example, both use cases are separated by a parallel gateway, as shown in the Figure 8. The parallel gateway has not yet been introduced as a pattern; in this implementation, it visualizes the (almost) parallel flow of the two messages. 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.

In the right path (see Figure 8), 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 left 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 8, 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.

VII. 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 presented 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. For realizing a pattern language, a grammar will be developed and evaluated in in future work.

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. In future work, we will thus present additional patterns and grammar, including usage examples for them. We will also aim to refine our choreography pattern language and evaluate its additional benefit through a concrete implementation.

References

[1] 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.

[2] European GDPR, “Complete guide to GDPR compliance,” Online. Available: https://gdpr.eu/ [retrieved: 04, 2023].

[3] 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/SharedDocs/Veroeffentlichungen/DE/Meldung/2023/meldung_2023_03_03_Aktualisierung_VAIT.html [retrieved: 04, 2023].

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

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

[6] 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.

[7] 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: 04, 2023].

[8] 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.

[9] “Workflow and decision automation platform,” Nov 2021, Online. Available: https://camunda.com/ [retrieved: 04, 2023].

[10] 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: 04, 2023].

[11] 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: 04, 2023].

 ISBN: 978-1-68558-043-8

[12] M. Fowler and J. Lewis, “Microservices a definition of this new architectural term,” 2014, Online. Available: https://martinfowler.com/articles/microservices.html [retrieved: 04, 2023].
[13] L. Krause, *Microservices: Patterns and Applications: Designing fine-grained services by applying patterns*. Lucas Krause, 2015.
[14] 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.
[15] T. Allweyer, *Geschäftsprozessmanagement: Strategie, Entwurf, Implementierung, Controlling. - Business process management: strategy, design, implementation, controlling.* W3l GmbH, 2005.
[16] B. Rücker and J. Freund, *Praxishandbuch BPMN 2.0 - Practice Handbook BPMN 2.0.* Carl Hanser Verlag München Wien, 2014.
[17] G. Decker, O. Kopp, and A. Barros, *An Introduction to Service Choreographies*, vol. 50, no 2 ed. Information Technology, 2008.
[18] B. Rücker, “The Microservices Workflow Automation Cheat Sheet,” 2018, Online. Available: https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa[retrieved: 04, 2023].
[19] D. Mikalkinas, *Situation-aware Modelling and Execution of Choreography*. Stuttgart University, 2015.
[20] 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.
[21] 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: 05, 2023].
[22] C. Richardson, “Eventuate Framework,” 2021, Online. Available: https://eventuate.io/[retrieved: 04, 2023].
[23] ——, “Eventuate Tram,” 2021, Online. Available: https://eventuate.io/abouteventuatetram.html[retrieved: 04, 2023].
[24] ——, “Eventuate Local,” 2023, Online. Available: https://github.com/eventuate-local/eventuate-local[retrieved: 04, 2023].
[25] J. Ladleif, A. von Weltzien, and M. Weske, “chor-js: A modeling framework for bpmn 2.0 choreography diagrams,” 2019.
[26] 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: 05, 2023].
[27] C. Chen, “Choreography vs orchestration,” Online. Available: https://medium.com/ingeniouslysimple/choreography-vs-orchestration-a6f21cfaccae [retrieved: 04, 2023].
[28] B. Rücker, “The Microservices Workflow Automation Cheat Sheet,” Online. Available: https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa [retrieved: 04, 2023].
[29] ——, *Practical Process Automation - Orchestration and Integration in Microservices and Cloud Native Architectures*. O’Reilly, 2021.

 ISBN: 978-1-68558-043-8