RESTFUL CHOREOGRAPHIES

ADRIATIK NIKAJ

BUSINESS PROCESS TECHNOLOGY GROUP
HASSO PLATTNER INSTITUTE
DIGITAL ENGINEERING FACULTY
UNIVERSITY OF POTSDAM
POTSDAM, GERMANY

DISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES EINES
"DOCTOR RERUM NATURALIUM"
– DR. RER. NAT. –

DATE OF DEFENSE: 15/11/2019

June, 2019

Supervisor: Prof. Dr. Mathias Weske, University of Potsdam
Reviewers: Prof. Dr. Gregor Engels, University of Paderborn, and
Prof. Dr. Cesare Pautasso, University of Lugano (USI)

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-43890
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-438903

ABSTRACT

Business process management has become a key instrument to organize work as many companies represent their operations in business process models. Recently, business process choreography diagrams have been introduced as part of the Business Process Model and Notation standard to represent interactions between business processes, run by different partners. When it comes to the interactions between services on the Web, Representational State Transfer (REST) is one of the primary architectural styles employed by web services today. Ideally, the RESTful interactions between participants should implement the interactions defined at the business choreography level.

The problem, however, is the conceptual gap between the business process choreography diagrams and RESTful interactions. Choreography diagrams, on the one hand, are modeled from business domain experts with the purpose of capturing, communicating and, ideally, driving the business interactions. RESTful interactions, on the other hand, depend on RESTful interfaces that are designed by web engineers with the purpose of facilitating the interaction between participants on the internet. In most cases however, business domain experts are unaware of the technology behind web service interfaces and web engineers tend to overlook the overall business goals of web services. While there is considerable work on using process models during process implementation, there is little work on using choreography models to implement interactions between business processes. This thesis addresses this research gap by raising the following research question: How to close the conceptual gap between business process choreographies and RESTful interactions? This thesis offers several research contributions that jointly answer the research question.

The main research contribution is the design of a language that captures RESTful interactions between participants—RESTful choreography modeling language. Formal completeness properties (with respect to REST) are introduced to validate its instances, called RESTful choreographies. A systematic semi-automatic method for deriving RESTful choreographies from business process choreographies is proposed. The method employs natural language processing techniques to translate business interactions into RESTful interactions. The effectiveness of the approach is shown by developing a prototypical tool that evaluates the derivation method over a large number of choreography models.

In addition, the thesis proposes solutions towards implementing RESTful choreographies. In particular, two RESTful service specifications are introduced for aiding, respectively, the execution of choreographies' exclusive gateways and the guidance of RESTful interactions.

ZUSAMMENFASSUNG

Das Prozessmanagement hat sich zu einer wichtigen Methode zur Organisation von Arbeitsabläufen entwickelt, sodass viele Unternehmen ihre Tätigkeiten mittlerweile in Prozessmodellen darstellen. Unlängst wurden zudem im Kontext der Business Process Model and Notation Choreographiediagramme eingeführt, um Interaktionen zwischen Prozessen verschiedener Partner zu beschreiben. Im Web nutzen interagierende Dienste heutzutage den Representational State Transfer (REST) als primären Architekturstil. Idealerweise implementieren die REST-Interaktionen der Dienste also die Interaktionen, die im Choreographiediagramm definiert wurden.

Allerdings besteht zwischen Choreographiediagrammen und REST-Interaktionen eine konzeptuelle Diskrepanz. Auf der einen Seite werden Choreographiediagramme von Domänenexperten mit dem Ziel modelliert, die Interaktionen zu erfassen, zu kommunizieren und, idealerweise, voranzutreiben. Auf der anderen Seite sind REST-Interaktionen abhängig von REST-Schnittstellen, welche von Web-Entwicklern mit dem Ziel entworfen werden, Interaktionen zwischen Diensten im Internet zu erleichtern. In den meisten Fällen sind sich Domänenexperten jedoch der Technologien, die Web-Schnittstellen zu Grunde liegen, nicht bewusst, wohingegen Web-Entwickler die Unternehmensziele der Web-Dienste nicht kennen. Während es umfangreiche Arbeiten zur Implementierung von Prozessmodellen gibt, existieren nur wenige Untersuchungen zur Implementierung von interagierenden Prozessen auf Basis von Choreographiemodellen. Die vorliegende Dissertation adressiert diese Forschungslücke, indem sie die folgende Forschungsfrage aufwirft: Wie kann die konzeptuelle Diskrepanz zwischen Choreographiediagrammen und REST-Interaktionen beseitigt werden? Somit enthält diese Arbeit mehrere Forschungsbeiträge, um diese Frage zu adressieren.

Der primäre Beitrag besteht in dem Design einer Modellierungssprache, um REST-Interaktionen zwischen Diensten zu erfassen—der RESTful Choreography Modeling Language. Formale Vollständigkeitseigenschaften (in Bezug auf REST) werden eingeführt, um Instanzen dieser Modelle, sogennante REST-Choreographien, zu validieren. Ferner wird eine systematische, halb-automatische Methode vorgestellt, um REST-Choreographien von Choreographiediagrammen abzuleiten. Diese Methode setzt Techniken des Natural Language Processing ein, um Interaktionen in REST-Interaktionen zu übersetzen. Die Wirksamkeit des Ansatzes wird durch die Entwicklung eines prototypischen Werkzeugs demonstriert, welches die Ableitungsmethode anhand einer großen Anzahl von Choreographiediagrammen evaluiert.

Darüber hinaus stellt diese Arbeit Lösungen zur Implementierung von REST-Choreographien bereit. Insbesondere werden zwei REST-Dienstspezifikationen vorgestellt, welche die korrekte Ausführung von exklusiven Gateways eines Choreographiediagramms und die Führung der REST-Interaktionen unterstützen.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- A. Nikaj, S. Mandal, C. Pautasso, and M.Weske. "From Choreography Diagrams to RESTful Interactions". In: *Service-Oriented Computing* ICSOC 2015 Workshops. Ed. by A. Norta, W. Gaaloul, G. R. Gangadharan, and H. K. Dam. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 3-14.isbn: 978-3-662-50539-7.

- A. Nikaj and M. Weske. "Formal Specification of RESTful Choreography Properties". In: Web Engineering. Ed. by A. Bozzon, P. Cudre-Maroux, and C. Pautasso. Cham: Springer International Publishing, 2016, pp. 365-372. isbn: 978-3-319-38791-8.

- A. Nikaj, F. Pittke, M. Weske, and J. Mendling. "Semi-automatic Derivation of RESTful Interactions from Choreography Diagrams". In: Enterprise, Business-Process and Information Systems Modeling. Ed. by R. Schmidt, W. Guedria, I. Bider, and S. Guerreiro. Cham: Springer International Publishing, 2016, pp. 141-156. isbn: 978-3-319-39429-9.

- A. Nikaj, K. Batoulis, and M. Weske. "REST-Enabled Decision Making in Business Process Choreographies". In: Service-Oriented Computing. Ed. by Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri. Cham: Springer International Publishing, 2016, pp. 547-554. isbn: 978-3-319-46295-0.

- A. Nikaj, M.Weske, and J. Mendling. "Semi-automatic derivation of RESTful choreographies from business process choreographies". In: Software & Systems Modeling 18.2 (Apr. 2019), pp. 1195-1208. issn: 1619-1374.

- A. Nikaj, M. Hewelt, and M.Weske. "Towards Implementing REST-Enabled Business Process Choreographies". In: Business Information Systems. Ed. by W. Abramowicz and A. Paschke. Cham: Springer International Publishing, 2018, pp. 223-235. isbn: 978-3-319-93931-5.

In addition to above publications as part of this thesis, I was also involved in the following research indirectly contributing to this thesis:

- S. Haarmann, K. Batoulis, A. Nikaj, and M. Weske. "DMN Decision Execution on the Ethereum Blockchain". In: Advanced Information Systems Engineering. Ed. by J. Krogstie and H. A. Reijers. Cham: Springer International Publishing, 2018, pp. 327-341. isbn: 978-3-319-91563-0.
- S. Haarmann, K. Batoulis, A. Nikaj, and M. Weske. "Executing Collaborative Decisions Con˛dentially On Blockchains". In: Accepted for Publication in BPM 2019 - Blockchain Forum. Springer International Publishing, 2019.

ACKNOWLEDGMENTS

I am very grateful to have written this dissertation under the supervision of Mathias Weske. Not only did he provide a clear guidance throughout this PhD journey, but he was actively involved in discussing with me the intricacies of business process choreographies from the very beginning. Moreover, I consider myself lucky to be part of his research group as I have spent the last 4 years working in a great scientific environment and gaining a wide range of experiences.

I would like to thank Gregor Engels and Cesare Pautasso for reviewing my thesis. Gregor Engels has played an important role in shaping my scientific research interests during my master's studies at his chair and has inspired me to follow the PhD studies. Cesare Pautasso is the main reason this thesis is about REST. I am very thankful to him for the early discussions that triggered the stream of research presented in this thesis.

I would like to express my gratitude to all the coauthors of our published research contributions: Sankalita Mandal, Cesare Pautasso, Fabian Pittke, Jan Mendling, Kimon Batoulis, Marcin Hewelt, and Stephan Haarmann. Special thanks go to Kimon and Stephan, Kimon for engaging in fruitful and enjoyable logical discourse and Stephan as a great sparring partner when it comes to quasi-philosophical discussion about choreographies and blockchains.

I thank all my colleagues for making BPT a great place to be and work; the colleagues who proofread my thesis Kimon Batoulis, Simon Remy, Sven Ihde, Sankalita Mandal, Stephan Haarmann, and Fabian Pittke; Jan Ladleif for designing the thesis cover; Kimon and Sankalita for sharing the difficulties of writing a PhD thesis.

Finally, I want to express my gratitude to my parents for their investment and love and in particular to my wife for her continuous support and push towards this major goal.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

API	application programming interface
ARS	accommodation reservation service
BPEL	Business Process Execution Language
BPM	business process management
BPMN	Business Process Model and Notation
BPMS	business process management system
CFP	call for papers
DMN	Decision Model and Notation
FEEL	Friendly Enough Expression Language
HATEOAS	hypermedia as the engine of the application state
MOOC	massive open online course
NLP	natural language processing
OCL	Object Constraint Language
REST	REpresentational State Transfer
RMS	review management system
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
WS-CDL	Web Services Choreography Description Language

Part I

BACKGROUND

1 INTRODUCTION

Today, enterprises interact with each other and their customers to bring common value to the market. Enabled by Internet and its trailing technologies, business process interactions around the world are made easier. Business process management (BPM) [99] is the discipline of modeling, analyzing, and executing business processes. To adopt the current trends in common value creation, BPM treats business to business interactions as first class citizens. To this end, business process choreography models are introduced as part of the Business Process Model and Notation (BPMN) standard [65]. They model the interacting processes between two or more business actors from a global perspective. Process choreography models serve as contracts that describe the message exchanges between participants and their behavior, in that who sends what messages in which order.

Business process choreographies originate from work on service interaction in the context of Service Oriented Architecture (SOA) [26] and Simple Object Access Protocol (SOAP) [100]. However, it is widely accepted that one of the most adopted paradigm for designing Web application programming interfaces (APIs) is REpresentational State Transfer (REST). Despite SOAP and REST being not comparable per se (the former is a protocol and the latter is an architectural style), comparison between SOAP- and REST-based solutions can be drawn [75, 107]. In this thesis, we investigate the conceptual relation between BPMN business process choreographies and RESTful interactions—interactions that obey to the REST principles.

Business process choreography models are designed by business domain experts to capture the interactions between the participants for reaching a specific mutual goal, e.g., booking an accommodation, organizing a scientific conference, or purchasing specific parts for manufacturing a certain product. Enacting a choreography model means to enable the individual participants to exchange the messages in the order defined by the choreography model. When it comes to RESTful interactions on the internet, Web engineers design REST interfaces that facilitate the business interaction specified by the choreography model. However, there is a conceptual gap between process choreography models and their RESTful interactions. Business domain experts are mostly not aware of the underlying information technology (IT) infrastructure and Web engineers tend to overlook the overall business goals of the Web services, leading to a misalignment between the two. This constitutes a problem as business processes are an important vessel for the overall alignment between the business strategy and IT [88].

While there is work on investigating the relation between BPMN and REST [30, 76, 77, 78, 103], there exists a research gap in enacting business process choreographies as RESTful interactions. Addressing this gap is very relevant because BPMN process choreographies are the standard language, when it comes to modeling business interactions, and REST is the dominant architectural style, when it comes to executing business interactions on the Web.

1.1 RESEARCH GOAL

This thesis' goal is to address the following main research question: How to enact business process choreography models as RESTful interactions on the Web? To address this question we decompose it into questions of finer granularity that aim at several relevant aspects. Eventually, we derive a set of objectives that, when fulfilled, jointly address the main goal:

- The first objective is about designing a language for capturing RESTful interactions into models as well as be conceptually close to business process choreography language. This language shall serve as a conceptual bridge between process choreography models and RESTful interaction models.
- The second objective revolves around designing an automatic method for deriving RESTful interaction models from business processes choreography models. The automation is required because we want to keep a clear separation of concerns between the domain experts and Web engineers.
- The third objective is to design completeness properties, with respect to REST, for validating the behavior of RESTful interaction models.
- The fourth and final objective is to design solutions that facilitate the enactment of RESTful interaction models.

1.2 CONTRIBUTIONS

This thesis contributes with several artifacts that are designed for reaching the research objectives listed above. These contributions collectively serve the thesis goal. Nevertheless, some contributions can be used as exaptation for similar problem spaces (e.g., implementing choreographies on the blockchain [54]). The list of this thesis' main contributions is provided below:

RESTFUL CHOREOGRAPHIES. RESTful choreography modeling language is introduced for capturing RESTful interactions into

models. RESTful choreography models, or simply RESTful choreographies, constitute the main contribution of this thesis (hence, the thesis title) as they are our prime designed artifact for aligning business process choreographies to their RESTful execution.

SEMI-AUTOMATIC DERIVATION. A systematic method is introduced to derive RESTful choreographies from process choreographies in a semi-automatic fashion. This is achieved by employing natural language processing techniques that harvest the business information embedded in the process choreography to generate REST interfaces. This preservers the separation of concerns between business domain experts and Web engineers. This contribution addresses the second research objective.

COMPLETENESS PROPERTIES. REST-specific formal properties are introduced to guarantee the lack of REST-induced deadlocks in the RESTful choreographies that are derived from correct process choreographies. These properties, if satisfied, render a RESTful choreography complete with respect to REST constraints. This contribution addresses the third research objective.

RESTFUL DECISION SERVICE. A RESTful-decision service is introduced to facilitate the implementation of RESTful choreographies. It contributes towards the fourth research objective stated above and it is concerned specifically with the execution of the choreographies' peculiar exclusive gateways.

CHOREOGUIDE. The RESTful choreography guide (ChoreoGuide) also contributes to the fourth research objective to facilitate the implementation of RESTful choreographies. It represents a central RESTful service that takes the participants "by the hand" and guides them towards a successful RESTful choreography execution.

1.3 STRUCTURE OF THE THESIS

This thesis is structured in four parts, shown as follows:

PART I. This introduction chapter provides the context, problem, and the proposed contributions treated in this thesis. Chapter 2 presents the main concepts that are necessary for following the rest of the thesis. Chapter 3 concludes this part by listing relevant related work that helps the positioning of this thesis' contribution in the body of existing knowledge in this particular research area.

PART II. This part constitutes the first main part of the thesis. The research questions and requirements are specified in Chapter 4. The thesis' big picture is first introduced in this chapter and later updated with the actual contributions on the consecutive main

chapters. The rest of this part is dedicated exclusively to the specification (Chapter 5), derivation (Chapter 6), and completeness properties (Chapter 7) of RESTful choreographies.

PART III. This part constitutes the second main part of the thesis and it focuses on facilitating the enactment of RESTful choreographies. The concept of RESTful decision service is presented in Chapter 8, while ChoreoGuide is introduced in Chapter 9.

PART IV. The last part of this thesis provides a comprehensive evaluation of deriving RESTful choreographies from business process choreographies in Chapter 10 before concluding the thesis in Chapter 11.

2 FOUNDATIONS

This chapter presents existing main concepts that set the stage for introducing our main contribution in the consecutive chapters. Due to business process choreographies being the major concept that this thesis is revolved around, this chapter motivates and elaborates concepts through the lens of business to business interaction or inter-organizational setting. This helps the reader to always see the relevance of the concepts at hand.

Since this thesis is situated in the area of Business Process Management (BPM) and Service Oriented Architecture (SOA), a short description of BPM and its lifecycle is provided in Section 2.1 followed by an introduction to service oriented architecture in Section 2.2. Section 2.3 introduces the BPMN business process model and collaboration model before Section 2.4 provides an overview of BPMN business process choreographies. For checking formal properties for some of our solutions, Petri nets are introduced in Section 2.5. The other main concept treated in this thesis is REpresentational State Transfer (REST) architectural style, which is introduced in Section 2.6.

2.1 BUSINESS PROCESS MANAGEMENT

The ultimate goal of any enterprise is exchanging value with its customers and business partners. To achieve this goal, an enterprise has to perform: core activities, which directly impact the creation of the value proposition [70]; and supporting activities, which are performed to improve the quality of customer service, delivery channels, relations with business partners etc. Whether in an organizational or technical setting, enterprises perform these activities to reach intermediate or end goals. The activities and the order they are performed constitute the enterprise's business processes [99]. The discipline studying business processes is Business Process Management (BPM). BPM is concerned with several aspects of business processes. Specifically it encompasses the design, configuration, enactment and analysis of business processes [93].

BPM lifecycle represents (see Figure 1) an overview of the creation and evolution of business processes inside an organization. The evolution of business processes (once these are identified and modeled) goes through the cycle of analysis, configuration, enactment, evaluation and redesign. Although there is a logical flow to the cycle phases, they can be visited independently where possible.

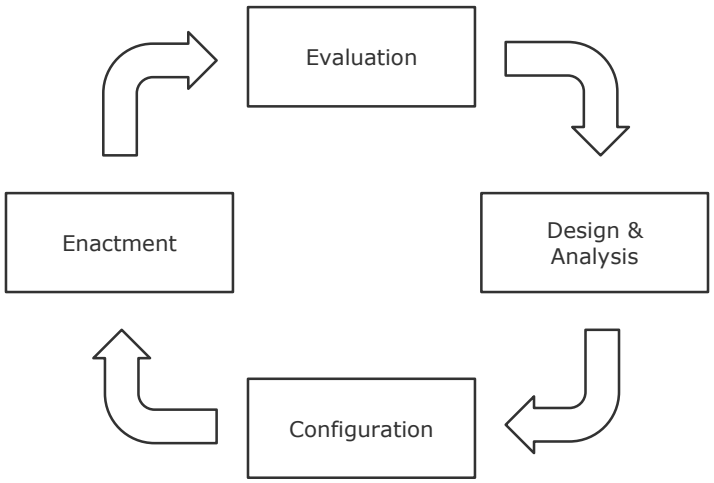

Figure 1: The BPM lifecycle [99]

DESIGN AND ANALYSIS. This phase is about identifying occurring processes in the organization and capturing them into models. Business process models are used as a mean of communication among process stakeholders about the as-is processes that are present in an enterprise. By analyzing them, important information can be derived about the state of the running processes in an organization, e.g., the existence of deadlocks, unfinished or unreachable activities, or activities that take longer than expected. Depending on the type of analysis, like structure or behavioral analysis, efforts of different complexity can be applied (ranging from simple validation with domain experts to verification of formal properties to very complex simulations) to retrieve insights about possible process improvement in terms of cost, efficiency and effectiveness. The improvement information is incorporated in the re-design phase of the process model.

CONFIGURATION. In the presence of a business process management system (BPMS), business processes need to be configured before being enacted. Depending on the BPMS technology, the process models are enriched adequately with technical details that allow their deployment and execution. This is an important phase because even well modeled business processes can be subject to bad deployment and execution due to wrong configuration.

ENACTMENT. During this phases, business processes are initiated and run continuously to serve their business goal. Their executions spans from fully-automatized scripts to process tasks that require users inputs. Depending on the BPMS features, different types of stakeholders can monitor the execution of the processes for different purposes. Such purposes include but are not limited to: having an overview of the state of particular instances; being prompted to enabled activities that need to be executed; detecting anomalies that need intervention like

activities that are taking too long to be finished; or, having an overview of the process resources and their distribution. The latter is a typical subject to the field of operational management [87]. In case of processes that interact with other organizations (business partners or costumers), their execution is also dependent from them.

EVALUATION. In this last phase of the BPM lifecycle, process models undergo scrutiny from process experts. By monitoring and analyzing execution information (usually captured into so-called process logs), business process experts evaluate the process quality based on certain metrics. Process mining [92] is a prominent research area that is concerned with the discovery, conformance and improvement of process models based on real execution logs. The outputs of the evaluation phase are inputs to the recurring cycle starting with the phase of process re-design and analysis.

2.2 SERVICE ORIENTED ARCHITECTURE

SOA is a software design paradigm that is centered around the concept of service—an independent software entity that satisfies a specific business goal. A service has four properties [26]:

- it represents a business activity that has a specific purpose, e.g., submit payment, request reservation.
- it is self-contained.
- it maybe composed of other services to offer complex business solutions.
- it is seen as a black-box from the service requester perspective.

Figure 2 depicts the three main roles of SOA. The service provider role represents companies or other business actors that design services and publish them on the service market. The service repository represents a service market or broker where services are published and made available for discovery by service requesters. The role of a service requester encompasses business actors that have specific business needs, which can be satisfied by certain services. Once the service requester has discovered the right service, it binds to the service provider in order to invoke the service. SOA is a design paradigm and does not make any assumptions on the Web technology used to implement such interactions.

However, the binding of the service provider and requester can be made possible by employing Web services [1] that are based on SOAP [100] or REpresentational State Transfer (REST) [23]. The work by Pautasso in [75] provides an extensive framework for deciding on SOAP-based

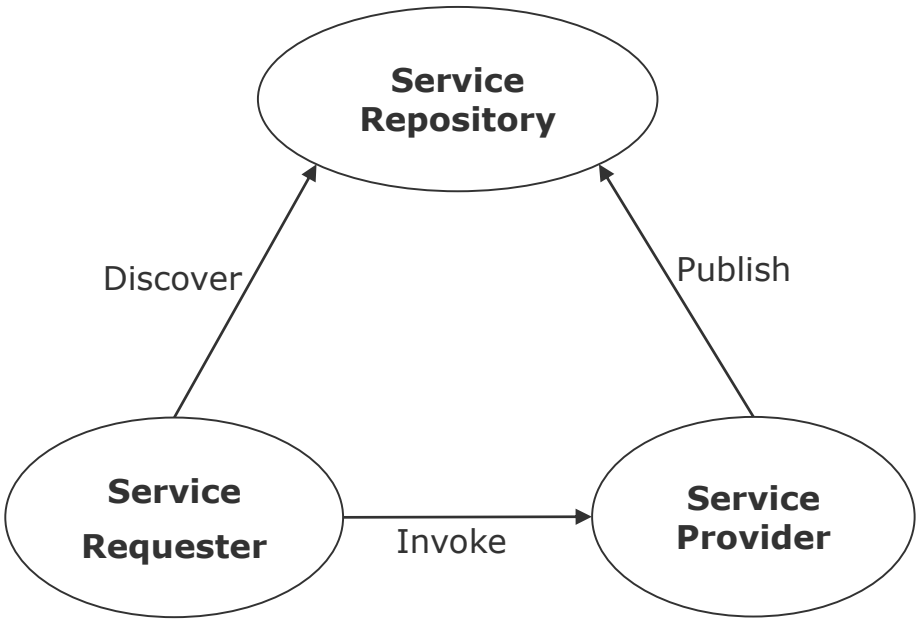

Figure 2: Service oriented architecture roles and their relations [26]

versus REST-based Web service solutions depending on specifics architectural choices. While there exist work in the SOAP-based implementation of business processes (see Chapter 3) in this thesis we investigate the REST-based implementation of business process choreographies. Hence, REST is described in details in the last section of this chapter.

2.3 BPMN BUSINESS PROCESS MODELS AND COLLABORATIONS

BPMN 2.0 [65] is the lingua franca for modeling business processes. In this section we provide a brief description of the BPMN 2.0 business process modeling language aided by an example of accommodation reservation service (ARS), which is inspired by airbnb[1]. Figure 3 models the process of dealing with an accommodation request that leads to either a successful reservation or to a failed reservation. The activities, events and gateways are ordered via sequence flows, which specify the causal relation between the source flow object and the target flow object (the target cannot start without the source being executed). Upon receiving a request for reservation (message start event), ARS checks automatically (script task) whether the guest request fits the host calendar, given the host has already provided his calender to ARS. In case of unavailability, ARS sends a *reservation not possible* message to the guest. Otherwise, it informs the host and the guest about the availability of the reservation.

1 www.airbnb.com

The action of sending a message is represented by the sent task (an activity with the filled message icon on the top left). The exclusive split of the control flow in the process is captured by the data-based exclusive gateway. This gateway validates the Boolean expressions of the outgoing branches, based on the process data, to decide which branch to activate. The parallel activities of informing the host and the guest are captured by a parallel block between the parallel split and the parallel join gateways. The former branches the control flow into different paths by activating all outgoing sequence flows while the latter signals the outgoing sequence flow only when all the incoming sequence flows are signaled.

The event-based gateway is similar to the data-based gateway, but the choice of the outgoing sequence flow is determined but external events and not by the local process data. Hence, this gateway must always be directly followed by catching intermediate events. In the example from Figure 3, once the host has been notified the process waits for three possible events to occur: the host does not respond for 24 hours and the reservation expires; a decline message from the host is received, which leads to informing back the guest about the unsuccessful reservation; or, the host accepts the reservation, the payment is processed and the confirmation is sent to the parties about the successful reservation. The end events capture the possible ways the process can come to an end.

Besides the activity-centric view of the process model, there are also constructs for describing the data that is read or written by the process activities. Data objects are the main artifact for capturing the data that is relevant to the business process. The state of the data (written in squared brackets at the bottom of the data object representation) provides a snapshot of the data objects at different points during the process execution, hence, describing how the data objects behave with the execution of the process model. An unidirectional or bidirectional association is used for connecting an activity with the data object (for the visual depiction refer to the legend in Figure 3).

Thus far, we described a set of basic elements that comprise an organization's business process model. However, an organization's end goal per se (value exchange) implies always the existence of business processes that communicate with the outside world. These processes, altogether, cover partially or fully the exchange of value and other interactions (with customers and business partners) that the enterprise undertakes. To capture these interactions, a process model is geared up with constructs that express the implicit or explicit information captured from or sent to the outside world.

First, in order to show the boundary of the organization with the outer environment the concept of pool is introduced. Collaboration models capture the interactions of at least two pools. A business process' sequence flow together with the flow objects they connect are contained inside a pool (a sequence flow cannot reach outside the pool).

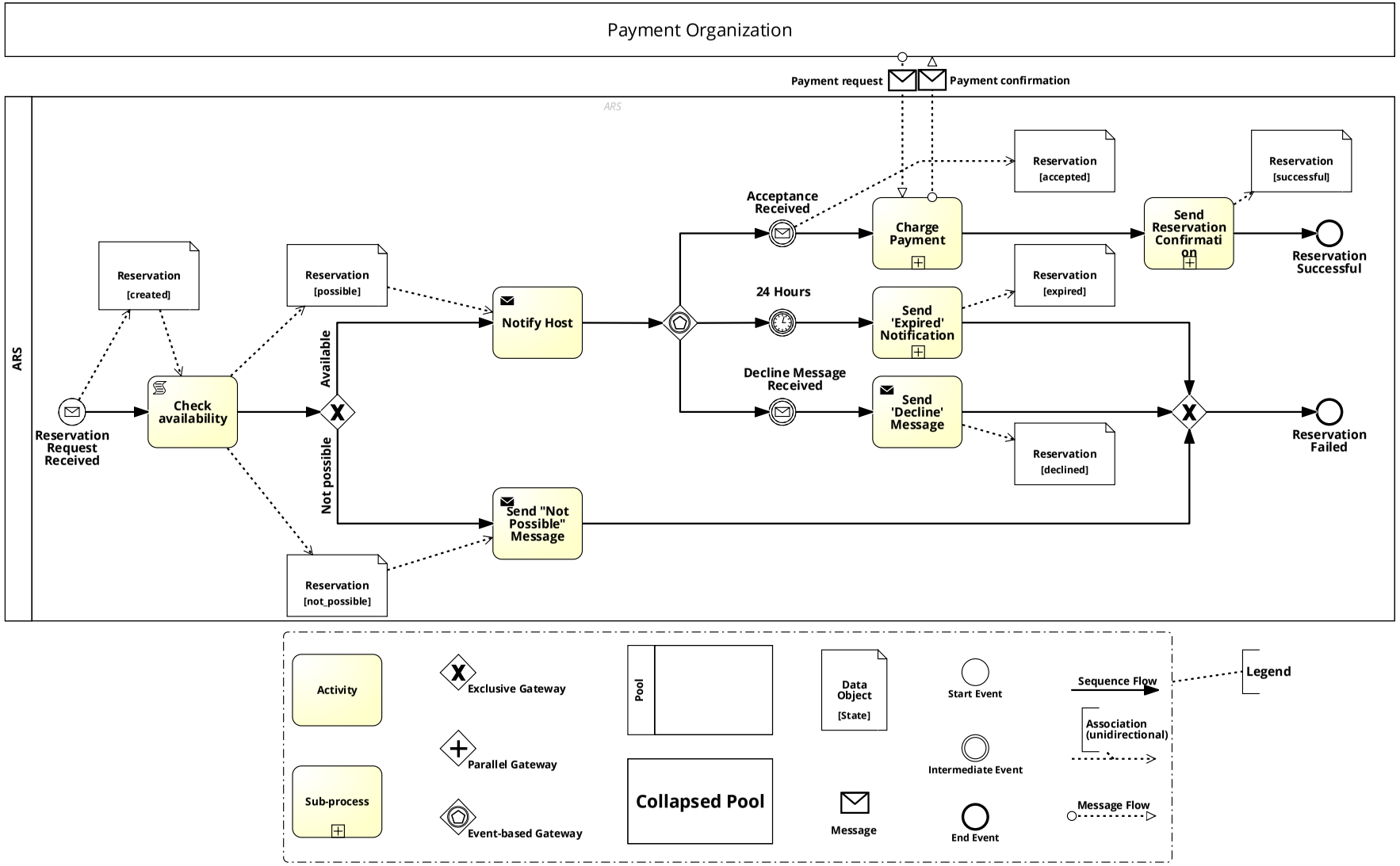

Figure 3: A BPMN business process collaboration model for ARS and its participants

Inversely, a pool does not have to show a business process, e.g., collapsed pools are pools that hide the internal business process model (black box) from the collaboration view, like the *Payment Organization* in Figure 3.

The flow objects that talk to the outside world are the events (catching and throwing) and the send/receive tasks. The explicit information flow between two participants is captured by the message flow, which connects: two pools; two flow objects belonging to separate pools respectively; or, a pool and a flow object from a different pool. The message flow refers to a single specific message. Graphically, a labeled message icon can be appended to the message flow (see Figure 3).

2.4 PROCESS CHOREOGRAPHIES

Business process choreography models, differently from the business process models, abstract from internal process activities and model the interaction between organizations from a global perspective with no bias towards any of the involved participants. The perspective, in this case, is that of an absolute observer that is aware of all the interactions that take place between the participants. The observer, however, is not aware of the activities performed locally by each participant. The participants, on the other hand, are only aware of the interactions in which they partake. Nevertheless, the actuation of the choreography model depends solely on the participants actuating their interaction activities in coordination as prescribed in the choreography model. In short, when it comes to deriving an implementation model from a choreography model, there is a paradigm shift from the global view to the local view. This paradigm shift makes the implementation of the choreography models rather complex.

Before we describe the BPMN 2.0 choreography modeling standard, we provide a short list of languages which are designed to model business to business interactions. This helps the reader getting an overview of the historical reasons that lead to the creation of the business process choreography standard. A comprehensive comparison of the languages in the following list is provided in [17].

BPEL. Business Process Execution Language (BPEL) [25] focuses on the activities that are responsible for sending and receiving messages. These activities are ordered via different control patterns, However, BPEL does not provide a standard graphical notation like BPMN. It is, however, very good at message correlation.

WS-CDL [32] Web Services Choreography Description Language (WS-CDL) is an XML-based language that specifies the interaction between different services that interact to reach a common goal. WS-CDL provides a global view of the interaction where each service is treated the same and there is no central mechanism that orches-

trates the interactions. The main idea behind business process choreographies' tasks come from WS-CDL where each task represents either a request from the initiator or a request followed by a response.

BPEL^{light}. It [61] extends BPEL by abstracting from the WSDL information.

WSFL [41] stands for Web Services Flow Language and combined together with XLANG [89] lead to the creation of BPEL [25].

LET'S DANCE [104] A visual language tailored to business users. It does not provide technology specific information. Like business process choreography and WS-CDL, Let's Dance specifies the global interaction between services or participants.

IBPMN [13] is an extension of BPMN that orders the interaction between collapsed pools (hence abstracting from internal activities) using BPMN control flow syntax and events.

Some of the concepts introduced by these languages were later incorporated in the BPMN 2.0 choreography model [4]. In this thesis, we use the BPMN 2.0 choreography modeling language as a standard for modeling cross-organizational interactions.

Figure 4 provides the BPMN 2.0 choreography diagram of ARS, which corresponds to the collaboration diagram of ARS from Figure 3. As observed, the choreography diagram models the causality relationships between the participants' interactions from the perspective of a global observer, e.g., after the host accepts the reservation request, ARS sends the payment request to the payment organization. The main composing unit here is the choreography task. A single choreography task models a message that is sent from one participant to another and, optionally, the message that is replied as a response (see *Request reservation* task from Figure 4). Hence, it contains two participant bands, one message or two messages (in case of a reply), and a textual label that describes the interaction. To visually distinguish the initiating message from the response message the latter has to be depicted with a dark shade. In the same fashion, the initiating participant must have a white background and the recipient of the initiating message must have a shaded background. This is needed to distinguish the initiating message from the reply message or the initiating participant from the other participant in case when the messages are graphically not present in the diagram (which is allowed by the standard).

Similarly to the business process diagram, the choreography tasks can be sequenced together via sequence flows, the semantic of which stands for: the sequence flow target can only happen after the sequence flow source has happened. However, this statement is provided from a global perspective. It may happen that the initiator of the target

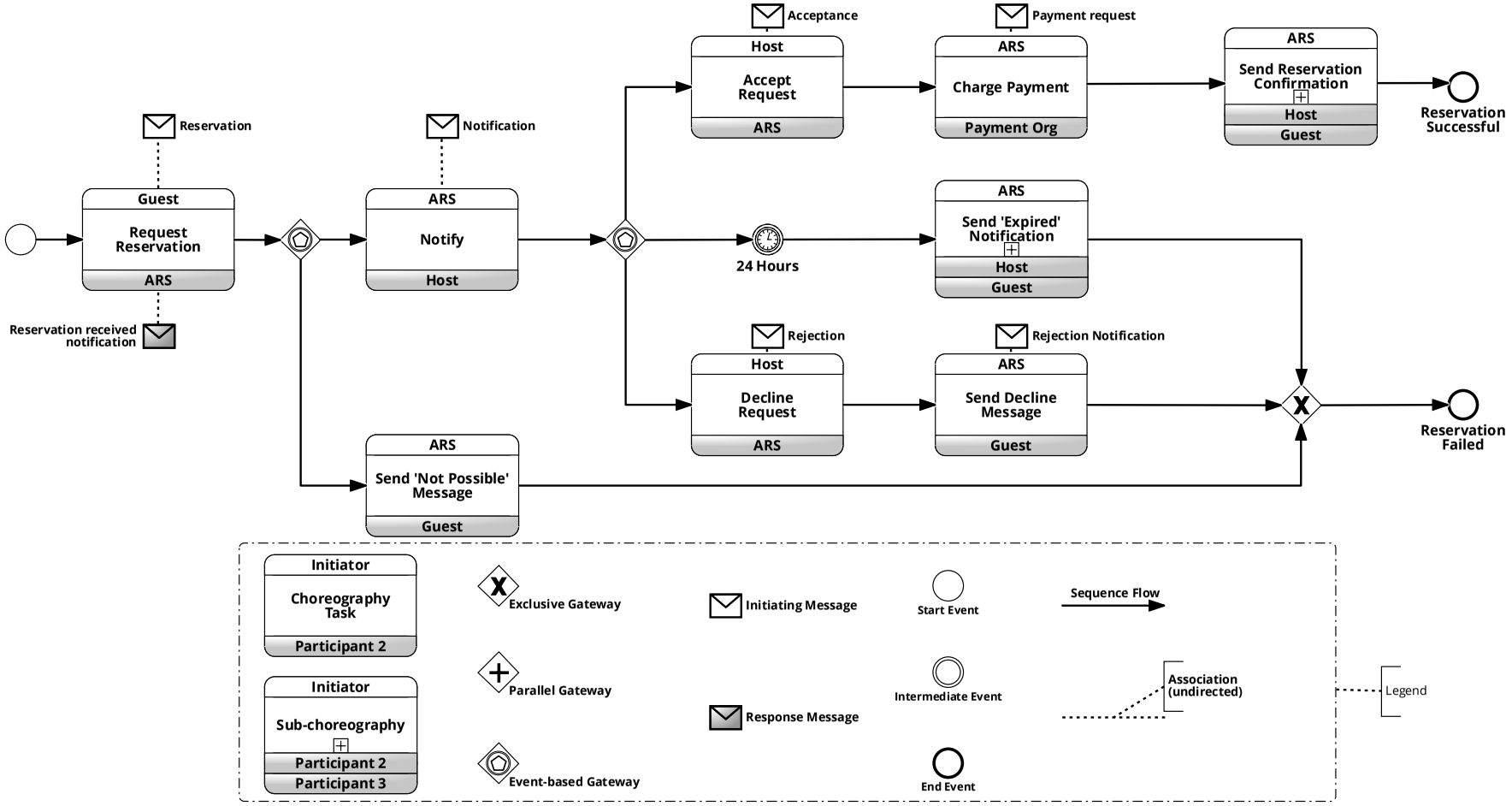

Figure 4: A BPMN business process choreography model of ARS, host, guest and payment organization

choreography task is not involved in the source choreography task and cannot possibly know whether the first task has been executed or not. In this case we say that such a choreography is not enforceable [105]. This is a fundamental difference between a choreography and an orchestration, where the process tasks connected by a sequence flow are actuated by a single entity that is aware of the state of the process at any given time and can enforce the order between its elements. To avoid designing an unenforceable choreography model the property of activity sequencing is introduced. The property states that for every choreography task, the initiator of the task must be involved in the directly preceding tasks. This must hold also in cases of sequence flow splits and joins. An exception are, of course, the choreography tasks that are at the beginning of the choreography diagram and have no other choreography tasks as predecessors.

In addition to sequence flows, business process choreography support other control flow patterns like parallel and exclusive control flows. As in business processes, control flow branching is expressed via gateways. However, the use of gateways in a choreography model comes with a set of limitations and assumptions. In this thesis we look at three most frequently used, from our experience, choreography gateways, i.e., parallel, exclusive and event-based gateways.

Parallel gateways split the control flow into two or more branches that are signaled simultaneously upon the gateway actuation. For an enforceable parallel split, the respective initiator of each directly following task must be a participant in all choreography task that directly precede the gateway. If there is no task that directly precedes the gateway but rather a chain of gateways, the choreography task that precedes the chain of gateways must follow the above constraint. The same must hold true for join parallel gateways. Figure 5 provides an excerpt from a choreography diagram with a parallel gateway and the corresponding business process collaboration diagram.

In choreography diagrams, exclusive gateways model alternative paths (see Figure 6). However, choreographies' exclusive gateways are constrained in their usage compared to their respective counterparts in business processes. In order for the choreography to be enforceable the following constraints should hold: The data used for the gateway conditions must have been in a message sent at some point in the choreography before the gateway. The message(s) containing the data is sent or received by all participants that are affected by the gateway and any change of the data must be visible to all these participants. And lastly, every participant must interpret the data in the same way. Applying these constraints to the example from Figure 6, ARS must include in the *Reservation Response* message the decision of asking for the payment or requesting additional details. The guest should be able to understand such decision and proceed accordingly. Meanwhile, ARS should not change his decision.

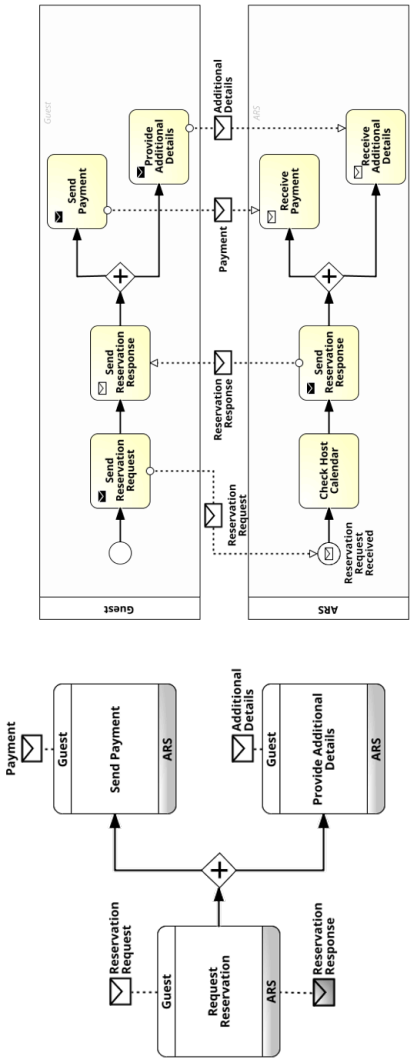

Figure 5: An example of a choreography parallel gateway and its corresponding collaboration diagram

However, there is a problem in implementing such constraints when it comes to the interaction of participants whose role can be filled by many possible business actors. The problem consists in that these different business actors can have different understandings of the data used for the decision making, leading the choreography to be out of synchronization. This is due to the fact that the business actors might be very diverse in terms of, e.g., domain and country. To ensure the enforceability of the choreography, we need to go a level closer towards the implementation level. To this end, in Chapter 8 we propose an approach on how to enforce choreographies' exclusive gateways in a Web service setting that employs REST.

Event-based gateways, like data-based exclusive gateways, model alternative paths in a process choreography. However, differently from exclusive gateways, not all participants take a decision. Some partici-

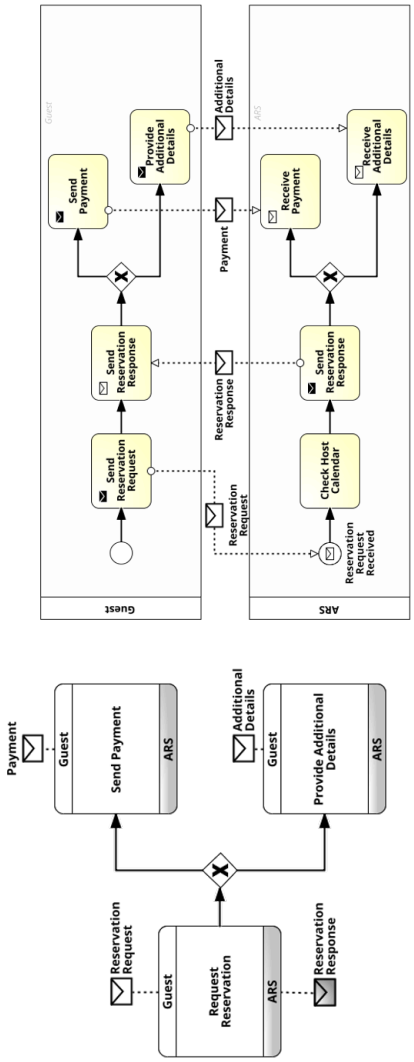

Figure 6: An example of a choreography exclusive gateway and its corresponding collaboration diagram

pants wait for exclusive messages to arrive, which determine the activation of the alternative paths. That means that at least one participant is not concerned with the decision making. From a collaboration perspective, the choreography event-based gateway is decomposed Figure 7.

Last, business process choreographies support events (like the timer intermediate event in Figure 4), but, compared to process models, they are limited in number and use. For a full list of allowed events in a BPMN choreography model, the reader is invited to consult chapter 10.5 in [65].

2.5 PETRI NETS

BPMN business processes are well-suited for modeling business processes at an abstraction level understandable by business process ex-

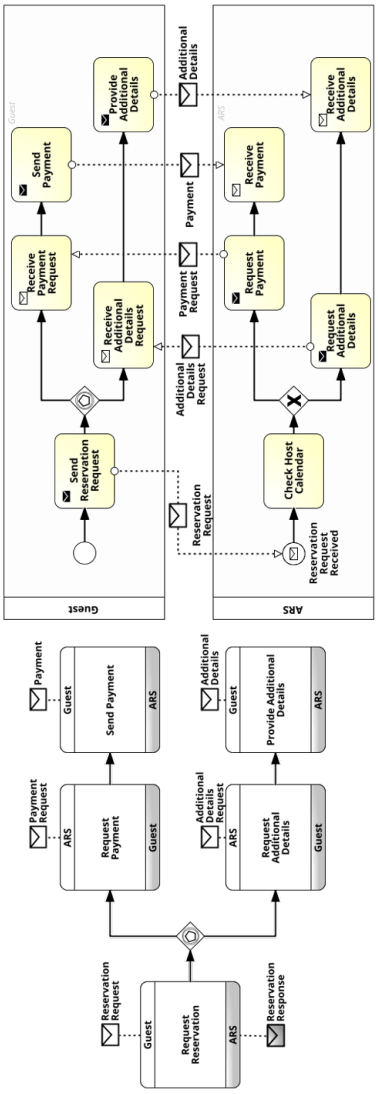

Figure 7: An example of a choreography event-based gateway and its corresponding collaboration diagram

perts. However, they lack proper formal semantics, hence, hampering their use for formal analysis. In order to apply formal analysis to business process choreographies several approaches can be followed, including: enriching business processes with more precise semantics; applying operational semantics; or, applying translational semantics. In this thesis we use the last approach.

Applying translational semantics means to translate one language to another by assigning new semantics with the aim of reaching a certain goal. In our case, the goal is to apply formal correctness check on business processes in general and business process choreographies in particular. In this thesis we use the translation of business process choreographies to Petri nets [80] for formally checking choreography

properties in Chapter 5. The use of Petri nets for formally analyzing workflows and business processes is well-researched [94, 95].

Petri nets are directed bipartite graphs that are abstract and unambiguous in their semantics.

Definition 2.1.
Petri net is a triple (P, T, F) where:

- P is a finite set of places
- T is a finite set of transitions
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations

◀

The Petri net in Figure 8 is derived from the ARS business process from Figure 3. The derivation is achieved by applying transformation rules from Dijkman et al. [45].

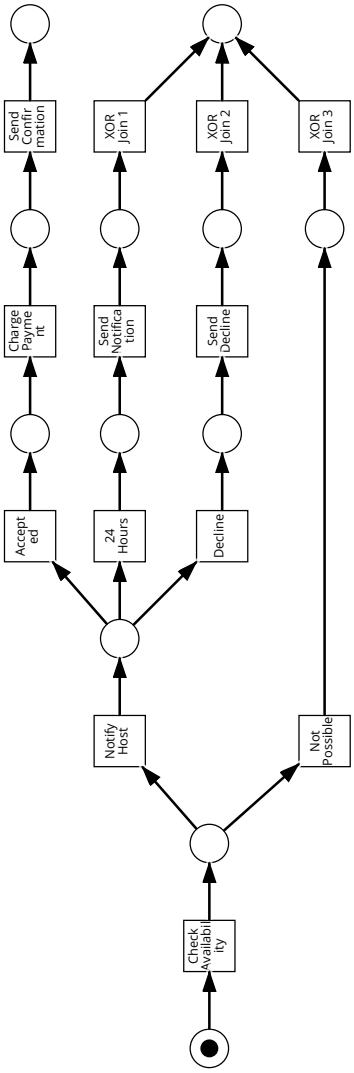

Figure 8: Petri net of the ARS business process

A place p is an input place of transition t iff $\exists f \in F \mid f = (p \times t)$. Correspondingly, a place p is an output place of transition t iff $\exists f \in F \mid f = (t \times p)$. The set of all input places and output places of transition t are denoted respectively with $\bullet t$ and $t\bullet$. In the same fashion we denote $p\bullet$ and $\bullet p$ the set of all transitions who have p as respectively an input place and output place.

Definition 2.2.
The marking of Petri net (P, T, F) is the function $M : P \to \mathbb{N}^0$ that assigns a natural number of tokens $n \in \mathbb{N}^0 = \{0, 1, 2, ...\}$ to places. ◀

The marking of the Petri net represent the state of the net at a given moment, i.e., the token distribution over the Petri net's places. The state of the net can change by firing transitions which displace tokens from input places to output places. More precisely, firing a transition $t \in T$ means consuming exactly one token from each place $p_1 \in \bullet t$ and produces exactly one token on each place $p_2 \in t\bullet$. This means that a transition cannot fire if a place $p \in \bullet t$ contains no token. If all input places contain at least one token the transition is enabled and can fire.

2.6 REPRESENTATIONAL STATE TRANSFER ARCHITECTURAL STYLE

The Representational State Transfer (REST) architectural style [23] is increasingly used for the development of Web services. Its architectural constraints contribute to among others, better scalability and portability. In virtually all cases, REST uses the HTTP protocol as a means of interactions between different participants.

REST consists of a set of five mandatory constraints plus an optional one. Fielding starts with an empty set of constraints (see Figure 9) and applies consecutively to the Web the following constrains:

Figure 9: Null-Style

1. **Client-Server**. This constraint ensures a separation of concerns between the user interface and the data storage (see Figure 10). This improves the portability (the client software component can run on different devices) and scalability due to the server side becoming less complex. In addition, the client and server components can evolve independently as long as the interaction interfaces do not change.

2. **Stateless**. This constraint requires from the clients to always send self-contained requests, i.e., each request must hold sufficient information for the server to understand it (see Figure 11). This

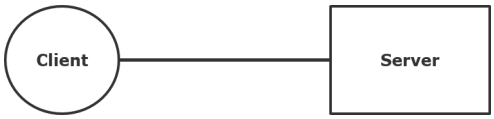

Figure 10: Client-Server

means that the server shall not hold any session state, but rather it is client's responsibility to keep the session state.

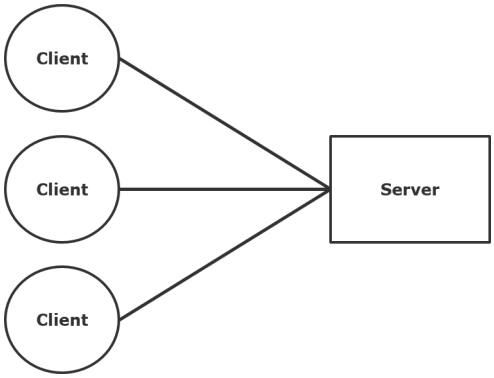

Figure 11: Stateless interaction

This constraint impacts positively the visibility, reliability and scalability properties. Visibility is improved because one does not need to look further than a single request to understand it. Likewise, it is easier to recover from partial failures, hence, improving reliability. Scalability is improved because the server does not invest resources for keeping the session state.

The trade-off of consist in that many repetitive requests may decrease the network performance because the server does not keep the session state and, therefore, reducing the amount of data sent on each request. Moreover, the server has less control over the state of the overall interaction because part of the control is given to the clients, who, might run in different machines and under different circumstances.

3. **Cache**. This constraint allows for responses to be marked as cacheable or not. In Figure 12, the client cache is represented by the small circle inside the client. Cacheable responses can be reused by the clients for the same requests without the need to resend them to the server. Thus, some interactions with the server are totally avoided.

   This leads to improved user-perceived performance (due to lower latency) and scalability. However, reliability is affected negatively because the cached response might not be up do date with the actual response.

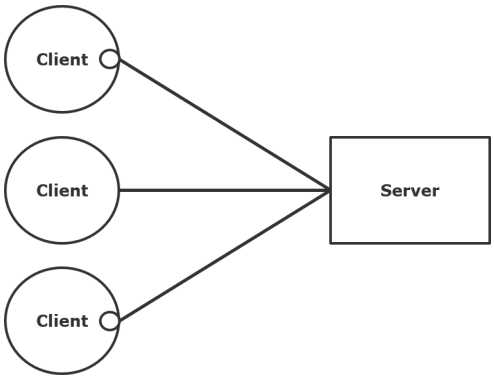

Figure 12: Cache

4. **Uniform Interface**. This constraint, which identifies REST architectural style, enforces the same interface between components on the Web (see Figure 13). The general interface simplifies the overall architecture with a cost on efficiency due to the generalization of, otherwise, specific requests. The uniform interface is realized by the following four constraints: identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine of the application state (HATEOAS).

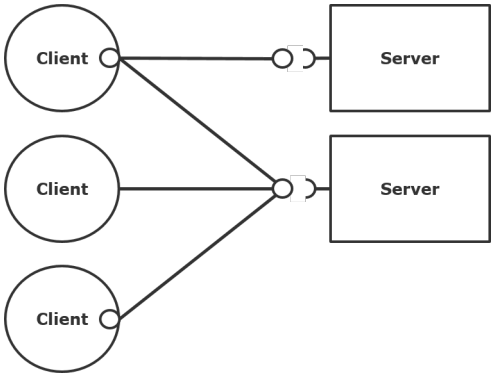

Figure 13: Uniform Interface

The resources resting on the server are globally and uniquely identified via a uniform resource identifier (URI). The interaction is achieved by using standard HTTP verbs (GET, POST, PUT, DELETE) on resources (see Table 1). Their state can be changed by the client through these REST verbs. Due to messages being self-descriptive, the server has sufficient information to process the messages. HATEOAS property enables clients to discover new resources by following URIs embedded into resources' bodies.

5. **Layered System**. This constraint allows for an hierarchical layer by limiting components' awareness to only the immediate layers that the components are interacting with (Figure 14). This allows

for a less complex system. Intermediaries can be used for load balancing or encapsulating legacy services. Due to the overhead added for each layer, the user-perceived performance is reduced.

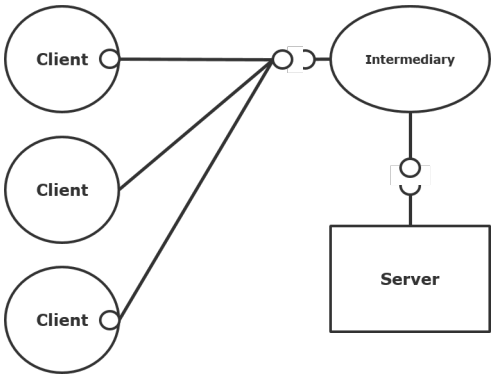

Figure 14: Layered System

6. **Code-On-Demand**. This constraint is about enabling clients to request and download executable code like scripts (Figure 15). On one hand this improves the system extensiblity. On the other hand, it reduces visibility as the server has no control over the execution of the application on the client side. Hence, this constrains is the only optional constraint in REST.

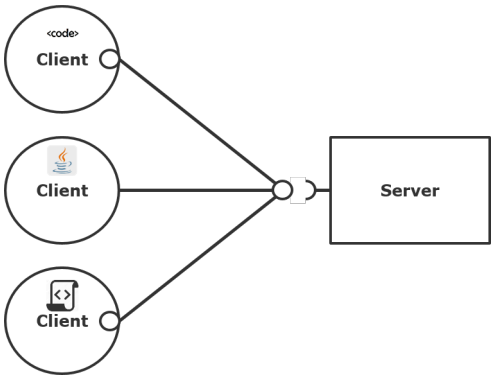

Figure 15: Code-On-Demand

To summarize, when two participants engage in a RESTful interaction they have to position themselves in the roles of client and server. A client is always the initiator of the interaction and the server follows always with a reply. The request, sent to the server, is self-contained, i.e., it contains sufficient information for the server to process it. The client uses a limited set of methods to access the resources that lie on the server side. The client uses the REST verbs from Table 1 to create a resource, read its latest state, update or delete it. This is done by using resource representations and not having direct access to the resource itself.

Table 1: REST verbs

REST Verb	**Description**	**Request and response example**
POST	Creates a new resource (not safe, not idempotent)	POST /reservations HTTP/1.1 HTTP/1.1 201 Created Location: /reservations/42
GET	Reads the representation of a resource (safe, idempotent)	GET /reservations/42 HTTP/1.1 HTTP/1.1 200 OK Hyperlink: /reservations/42/details
PUT	Updates an existing resource or creates a new one (not safe, idempotent)	PUT /reservations/42 HTTP/1.1 HTTP/1.1 200 OK Hyperlink: /reservations/42
DELETE	Removes a resource (not safe, idempotent)	DELETE /reservations/42 HTTP/1.1 HTTP/1.1 204 No Content

With every reply, the server presents the client with possible future interactions depending on the goal of the interaction. Future interactions are provided in the form of hyperlinks, which represent URIs of different resources that clients can follow to progress the overall interaction. For example, the guest (client) sends a request for creating a new reservation on ARS (server). If the reservation is accepted ARS replies with a link for paying the reservation. Otherwise, it replies with a hyperlink that provides details of the rejected reservation.

There are cases when the server needs to notify the client about a change of the resource state. To avoid the server asking frequently about the updated state of the resource, other solutions have been investigated [77]. One example is that the client-server roles are switched so that the participant who played the role of the client can also provide a service as a server in order to get requests that inform him/her about the updated resources. However, easier solutions can be used,

i.e., emails. For example, ARS can send an email to the host informing him about the reservation request. The email content can provide hyperlinks that the host can follow to either reject or decline the request.

3

RELATED WORK

This chapter provides an overview of related work in the area of business process choreographies and their implementation. In addition, it highlights the positioning of this thesis' contribution in the body of current research by identifying and targeting the research gap in the said area. The individual contributions of related work presented in this section pertain to one of the following three groups (not necessarily mutually exclusive): process choreography implementation; modeling of RESTful interactions; and, BPM and REST. Following this categorization, the first group of related work considers behavioral analysis and implementation of process choreographies encompassing concepts like enforceability, deadlock freedom, instance correlation etc. The second group of related work is about capturing and modeling RESTful interactions (REST API) between services. In the third group, we zoom in at the solutions that investigate the BPM-REST relation in particular.

3.1 PROCESS CHOREOGRAPHY IMPLEMENTATION

In this section we look at related work that bridges the gap between choreography models and their implementation. As argued in Chapter 1, implementing choreographies translates to enabling the involved participants to execute correct message exchanges in the order prescribed by the choreography. Enabling, in this case, means to equip each individual participant with methods and tools that ensure a correct execution of the choreography.

First work on bridging the choreographies global view and the participants local view is provided by Zaha et al. in [105]. The authors use Let's Dance [104] (see Section 2.4) as a modeling languages for capturing process choreographies. They argue that a choreography model cannot be locally enforced unless certain properties are met on the global model. Specifically, if the sender of a message is not involved in the preceding interaction (either as a sender or receiver) the order of the interactions cannot be enforced locally because the sender is not aware of the preceding interaction occurrence. This property should hold true even in cases of control flow forks like exclusive, inclusive or parallel splits. For example in the previous chapter, the *Host* has the choice to accept or decline the reservation request. In order to do so, the *Host* has to be aware of the preceding activity. In the BPMN 2.0 standard [65] this property is referred to as the *activity sequencing property* (see Section 2.4).

The authors describe the property on example bases (concrete Let's Dance choreography models) and provide no formal definition nor automatic check. These are, however, addressed in [14] by Decker and Weske. The authors introduce interaction Petri nets to formally express and automatically check the enforceability property, nevertheless, not without introducing a trade-off. Interaction Petri nets, which is an extension of Petri nets (see Section 2.5), is a low-level language which provides formal correctness but it is not suited for business users, like Let's Dance or BPMN process choreographies are. Hence, a mapping from a high level choreography language to interaction Petri nets would be required.

In [15], BPEL4Chor is introduced as an alternative to WS-CDL [7] for capturing choreographies. BPEL4Chor is an extension of BPEL [25] (Business Process Execution Language, see Section 2.4), based on Simple Object Access Protocol (SOAP) [100]. It contributes in bridging choreographies and their implementation, albeit, upward wise, i.e., the choreography is described as a composition of orchestration-level constructs. The implementation of the choreography is achieved by executing all the individual services described in BPEL. BPEL4Chor does not provide a graphical notation, hence, making it difficult for non-technical users to have a holistic overview of the choreography.

To counter this drawback, the authors propose in [16] a derivation method that takes as input a BPMN collaboration model (see Section 2.4) and outputs a BPEL4Chor model. The mapping is not straightforward as the modeling paradigms differ utterly, i.e., BPMN is a graph-based language while BPEL is a block-based language. Several other related work focus on translating BPMN to BPEL, like the work in [72] and [71], which [16] extends upon. Similarly, Mendling et al. [48] show how BPEL process definitions can be derived from a global WS-CDL [31] model for each participant of the choreography. This derivation is fully-automatized for certain blocks and semi-automatized for those blocks whose context plays an important role. Likewise, transformations to BPEL also exist from other graph-based process modeling languages [49, 106].

Recently with the surge of blockchain technology [54], new possibilities for implementing process choreographies have emerged. One notable work that makes use of blockchain for enforcing and monitoring choreographies is presented in [97]. The authors use Ethereum blockchain [11] as an intermediary platform that enforces the execution of the choreography by employing smart contracts [69]—Turing-complete programs that run on the Ethereum blockchain. Using blockchain technology is particularly beneficial in an environment where there is a lack of trust between business partners. However, choreography diagrams intrinsically lack a central system (i.e. a system shared between the business participants) that manages the common data and enforces the interaction behaviour. These are rather properties of an orchestra-

tion setting, which a choreography language like BPMN process choreography does not capture. This requires a systematic approach to investigate the appropriateness of choreography languages for capturing the interactions that are performed via the blockchain. Despite some initial contributions on the relation between BPM and blockchains (like in [51]), we argue that work on the choreography-blockchain relation is still in its infancy. In comparison, this thesis proposes a hybrid model (semi-orchestration service that guides the choreography) for executing RESTful choreographies in the context of RESTful services, but an additional resource data model is used to capture complementary information to the choreography model, e.g., the shared data and conditions on their state (see Chapter 9). Similar work shows that blockchain can replace traditional shared data storage [102] in the context of interacting software components. Moreover, it is shown that using cloud services for orchestrating choreography diagrams is cheaper than using Ethereum blockchain [84].

Enforceability is a necessary condition for a correct choreography implementation but not a sufficient one. A key aspect for ensuring a correct process execution consist in the data exchanged between participants. The participants need to have a common understanding of the data they share during the enactment of the process choreography. In case the participants have already a system in place, there is the need for data integration. Meyer et al. [52] provide a model-driven approach for describing and automating the data exchange between business processes belonging to choreography's participants respectively (see business process collaboration in Section 2.4). The authors propose a global data model as an agreement between participants on the shared data. The global data model is then matched individually to each participant's local data model via schema matching. Nevertheless, despite having a common agreement on the global model, there is yet the possibility of wrong message content. E.g., the payed amount is not equal to the billed amount. This thesis addresses these types of problems in Chapter 9.

Engels et al. propose in [20] an approach to transform collaboration diagrams to Java [2]. Collaboration diagrams in UML 1.1 [62] (known as UML communication diagram in UML 2.0 [63]) are used to model the interaction between different objects (akin to sequence diagrams but with no timeline segments involved). Collaboration diagrams can be considered a choreography language because the modeling perspective is a global one. They model the messages between objects and the order the messages are sent. In this approach, the execution of the collaborations diagram as Java code is considered an orchestration of the interactions between the objects. In this thesis, we propose a semi-orchestration service (see Chapter 9) that guides the RESTful execution of a given choreography.

Lastly in this section, we cover related work on choreography instance correlation. Since a single choreography model is instantiated many times (otherwise, there is hardly the need to create the model), every participant needs to correlate the messages that belong to the same local instance. In [3], the authors provide a framework which consist of correlation patterns for grouping messages together in the context of service oriented architecture. They show that BPEL provides good support for most of the correlation patterns. The work on [12] provides a more formal approach for isolating interaction instances belonging to the same choreography instance. The authors introduce ν*-nets as a Petri net extension which captures names creation and passing. In addition, [52] addresses the correlation between messages in a BPMN collaboration diagram by correlating messages spawn from local process instances via the aforementioned (previous paragraph) global data model.

3.2 MODELING RESTFUL INTERACTIONS

In this section we consider related work that investigate different ways of capturing RESTful interactions into models. REST is an architectural style that is built upon restrictions, which simplify the interaction on the Web in exchange for portability, scalability, evolvability etc. Thereby, REST mainly defines what is not allowed rather than what is allowed. To counter that, scholars and industry members alike have proposed several languages that enables the service developers to correctly model and implement REST-compliant interactions.

Valverde and Pastor [90] introduce a REST metamodel to counter the lack of a formal specification for RESTful Services. The metamodel enables a model-driven engineering approach to derive machine-readable formats that implement RESTful services. Similarly, Laikorpi et al. [36] propose a step-wise procedure, consisting of model-to-model transformations, to arrive from a REST API specification to a software artifact. Schreier [85] provides a more extensive technology-agnostic REST metamodel that has a special focus on the behavioral aspect. The ultimate goal of proposed work is to provide a high-level abstraction language which can be transformed to low-level technical models. The drawback of these meta-models is that they have no graphical representations the REST API developers can make use of. Our approach uses an established BPMN standard to specify the RESTful interactions from a global perspective.

Haupt et al. [28], differently from the aforementioned approaches, uses UML sequence diagram [63] (a graphical language). The authors take a conversation based perspective for capturing interactions of a single clients with RESTful services. Several REST interacting patterns are modeled to demonstrate the appropriateness of the approach. In addition a model driven approach is sketched for deriving the services'

RESTful APIs. The limitation of this work is that it captures only the interaction of a single client with one ore more servers. However, one important aspects of Web services is connecting many clients (humans or machines) like Airbnb, Facebook, Easychair, Google etc. Another drawback is that UML sequence diagrams are not suited for branching interactions (parallel or exclusive).

Some of these approaches lose the information that holds the Web service business goal, e.g., whether a specific URI is intuitive enough and relays its purpose to the client. In this thesis, URI are used to convey the business goal of the interactions, hence, increasing the understandability and facilitating the development of client-side applications.

3.3 BPM AND REST

Since this thesis is situated between the area of BPM and REST, in this section we look particularly at related work covering the relation between the two. One of the earlier work, which provides a closer look into BPMN and REST, is that of Pautasso in [76]. The BPMN process diagram is enriched with a light-weight notation to express REST notions in conjunction with business processes. The author proposes several ways for combining processes and REST: process tasks can interact with external REST resources via REST methods; a process, a sub-processs, or a task can constitute a REST resource per se. Pautasso and Wilde [77] extend the previous work by offering a catalog of technical solutions for addressing the problem of *notification push* to the client. Therefore, clients can be notified whenever there is a task or process state change without having to send several GET requests before the new state is reached. Another work in the same direction is presented by Xu et al. in [103]. The authors, propose an architectural style where processes elements are considered as REST resources and are managed using REST methods. These contributions, however, do not cover concrete (formal) properties for checking the correctness of the RESTful interactions. In addition, they provide an activity-centric perspective where data objects are ignored. The choreography perspective is not taken into account as the notation is based on the business process level. In short, the main difference consists in that, in this thesis, we use choreography diagrams to describe or prescribe RESTful interactions, while the related work uses REST to allow the management of business process models and their instances in a Web setting.

Cesare et al. propose an extension of BPMN choreography diagrams to model RESTful conversations in [78]. The extension, named RESTalk, is elaborated further and evaluated in [30]. RESTalk is simplified to only focus at the interactions that the server conducts with its client. We argue that this setting is more akin to an orchestration setting from the server's local point of view since the proposed language models only a single server and its behavior with the client. Furthermore, the

labels, which describe the interaction, are omitted. In this thesis, we use the business information embedded in the choreography diagram, especially in the labels, to derive better REST APIs following quality guidelines. Moreover, we argue that modeling RESTful interfaces from a global perspective (no bias towards and REST server) is important for capturing the state of common resources and the allowed interactions via interfaces that are derived from a global logic (see Chapter 6).

Part II

RESTFUL CHOREOGRAPHY LANGUAGE

4

RESEARCH QUESTIONS AND REQUIREMENTS ANALYSIS

In this chapter, we focus on formulating the research questions that are addressed by this thesis. Consequently a set of requirements are specified for designing artifacts that (partially) solve the research questions. The requirements are, then, treated in the subsequent five chapters constituting this thesis' main contribution.

4.1 RESEARCH QUESTIONS

The observation is that, on the one hand, REST architectural style is widespread and is one of the most preferred styles among Web designer. On the other hand, business process choreography is the standard for modeling business to business interactions. Ideally, RESTful Web service interactions should implement the interactions defined at the business process choreography level, albeit, at a more technical level. This observation leads to the research question: How to close the conceptual gap between business process choreographies and RESTful interactions? We tackle this research question in a top-down fashion. Thus, the main research question is reformulated as follows: How to enact business process choreography models as RESTful interactions on the Web? The main research question (MRQ) is decomposed into three fine-grained research questions (RQ) that, when addressed properly, lead to the answer of the MRQ. This is visually represented in Figure 16.

MRQ. How to enact business process choreography models as RESTful interactions on the Web?

RQ1. How to capture RESTful interactions?

The MRQ revolves around two entities: business process choreography models and concrete RESTful interactions. The choreography models are situated at the MOF's M1 level [68] (see Figure 16) while RESTful interactions belong to the Mo level. To narrow down the conceptual gap, we first need to know how to capture RESTful interactions into models (M1 level). This enables us to argue on the relation between models that are on the same abstraction level.

RQ2. How to derive RESTful interaction models from business process choreographies in an automatic fashion?

Choreography diagrams, on one hand, are modeled from business domain experts with the purpose of capturing, communicating, and, ideally, driving the business interactions.

RESTful interactions models, on the other hand, reflect the RESTful interfaces that are designed by Web engineers with the purpose of facilitating the interactions between participants on the Web. In most cases, business domain experts are unaware of the technology behind Web service interfaces and Web engineers tend to overlook the overall business goals of Web services. We need to know how to transform the information that is provided in process choreographies into REST-specific information needed to design RESTful interactions models, while preserving the separation of concerns between the choreography modelers and the Web service engineers. That is why this research question emphasizes the automatic part of the derivation.

RQ3. How to facilitate the enactment of RESTful interaction models?

Enacting a RESTful interaction model means to make sure that the RESTful interactions between the participants are performed according to the specified model. Since participants are independent and act autonomously, it is important to build systems that guide the participants through correct interactions with respect to REST principles.

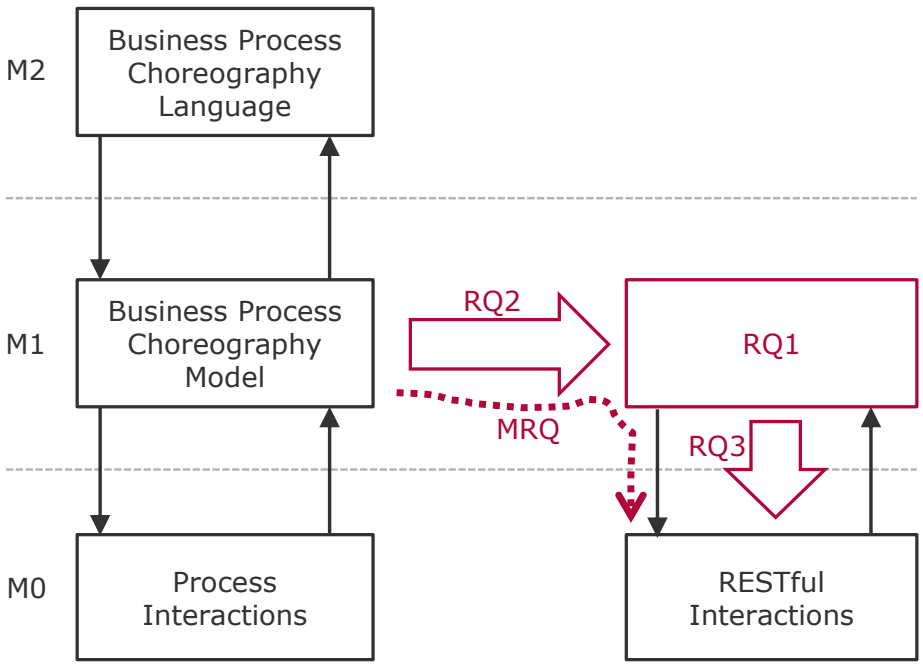

Figure 16: Overview of the research questions

4.2 REQUIREMENTS

By addressing these three research questions, we should be able to answer the main research question, at least partially. In sum, in order to derive RESTful interactions from process choreography models, the following artifacts are needed: models for capturing RESTful interactions; an automatic method for deriving these models; and, approaches that facilitate the enactment of such models. Hence for designing these artifacts, the following high level requirements are specified accordingly (see Figure 17):

R1. Design a modeling language for capturing RESTful interactions.

This requirement addresses the research question RQ1. In order to capture RESTful interactions in models, a language for designing said models needs to be specified at the meta-model level (M2 level). Hence, R1 is concerned with the design of a modeling language that expresses RESTful interactions, but also a language that is conceptually close to BPMN process choreography language.

R2. Design an automatic method for deriving RESTful interaction models from business process choreography models.

This requirement addresses the research question RQ2. Given a business process choreography model, we want to generate in an automatic fashion a RESTful interaction model that implements the choreography model. The automation is desired because we want to preserve a clear separation of concerns between the business process choreography modeler and the Web service engineer. Naturally, this derivation has to be specified at the language and model levels (M2 and M1 levels) (see Figure 17).

R3. Define completeness properties for RESTful interaction models.

Given a RESTful interaction model that can be derived automatically or designed manually, we have to make sure that the model is complete with respect to REST constraints. These properties should be defined at the language level (M2 level in Figure 17). This requirement addresses research question RQ1 because it makes sure that the RESTful interaction model is complete and correct, i.e., enforceable [99] and deadlock-free.

R4. Design a supporting system for enacting RESTful interaction models.

The enactment of the interactions is met with problems like misinterpretation of the data being exchanged or wrong order of interaction. Therefore a supporting system is needed for guiding the concrete RESTful interactions between the choreography participants. This requirement addresses research question RQ3.

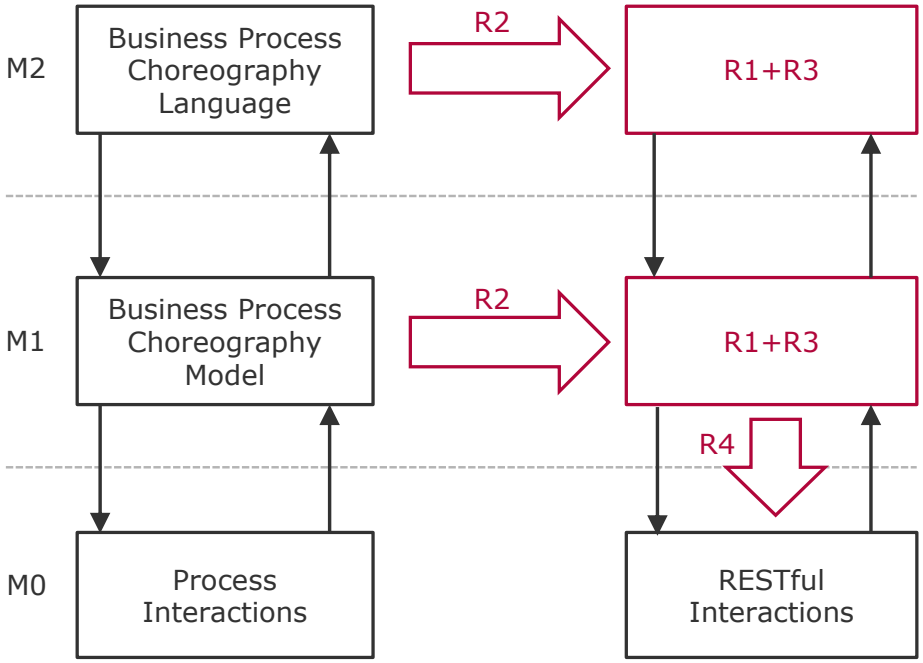

Figure 17: Overarching approach

Figure 17 depicts this thesis' overarching approach. The requirements are positions accordingly to the artifacts that need to be designed or specified. For the coming five chapters, each requirement is replaced with a specific artifact that fulfills that specific requirement. Hence, this figure reappears in an updated form at the beginning of each chapter to show how the chapter's specific contribution is positioned in the thesis' total contribution.

5

RESTFUL CHOREOGRAPHIES

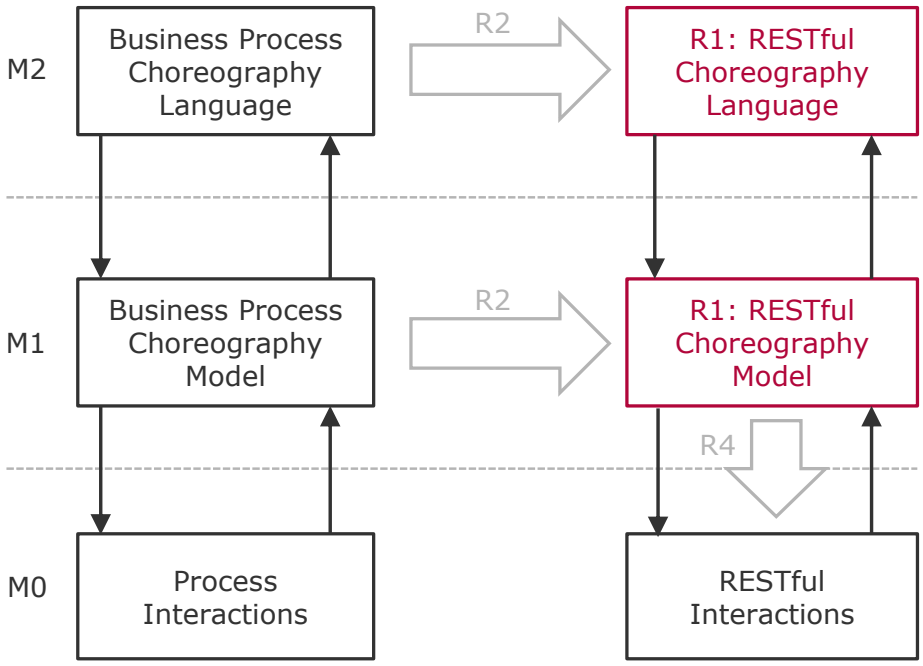

Figure 18: RESTful choreography language and RESTful choreographies

This chapter addresses requirement R1. As argued in the previous chapter, requirement R1 states the need of capturing RESTful interactions in RESTful interaction models. Hence, we specify a language that allows the design of said models—RESTful Choreography Language, the thesis' first main contribution. Throughout the rest of the thesis, we refer to RESTful interaction models as RESTful choreography models, or simply RESTful choreographies. RESTful interactions are instances of RESTful choreographies. First we provide the reasons behind the main design decision of specifying RESTful Choreographies Language as an extension of BPMN 2.0 Process Choreography Language. Then, we provide the formal specification of RESTful Choreography Language followed by the visual annotation. This chapter's running example is the *ARS* choreography which is introduced in Section 2.4 and depicted again for convenience (without the legend) in Figure 19. This chapter concludes with RESTful choreographies derivation guidelines and design patterns. The contributions presented here are partially based on [57], where RESTful choreographies are first introduced by Nikaj et al.

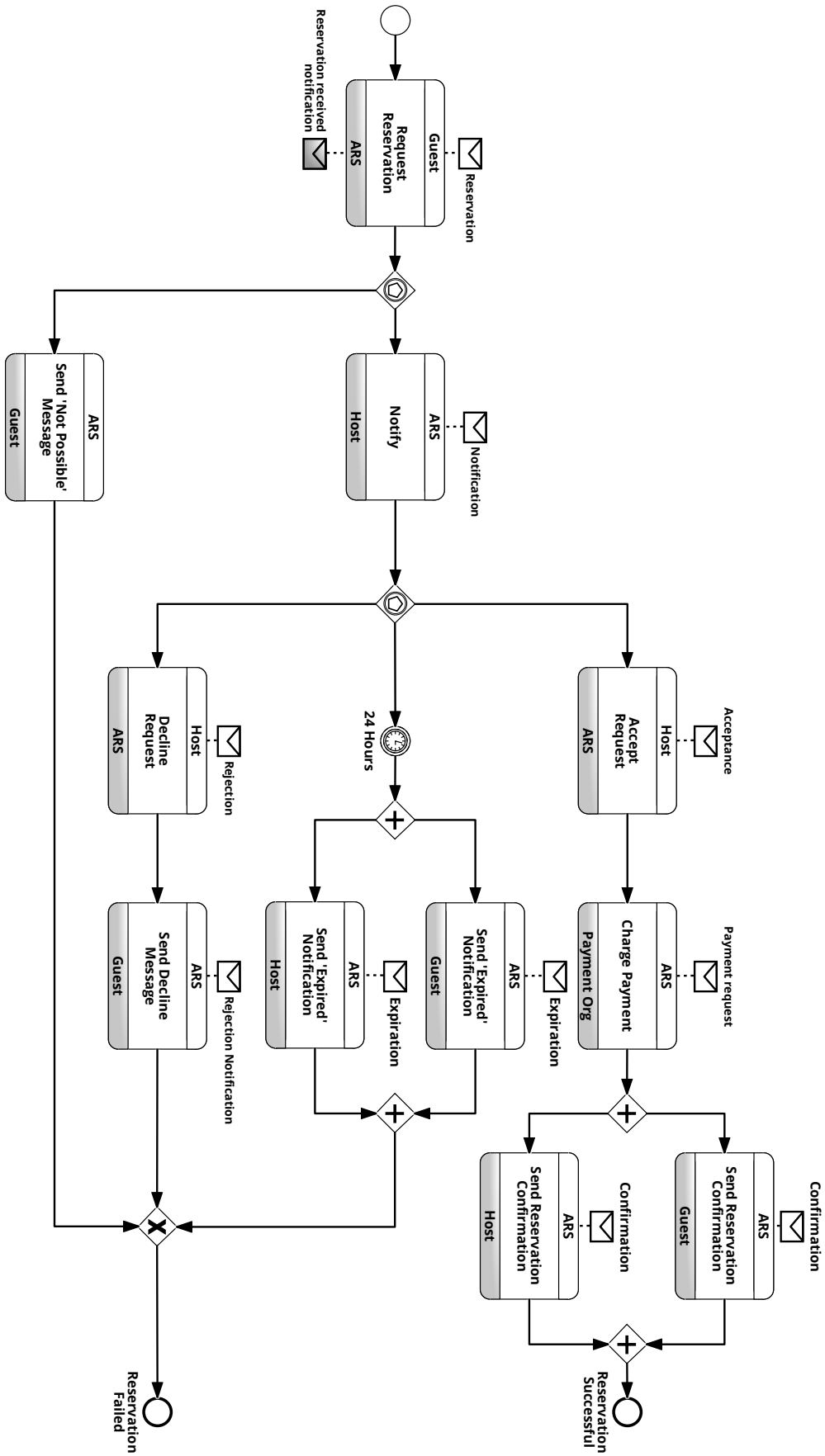

Figure 19: The choreography diagram of an *Accommodation Reservation Service (ARS)* without sub-choreographies

5.1 MAIN DESIGN DECISIONS

We discuss the main design choice behind extending the BPMN 2.0 Choreography Modeling Language [65] to capture RESTful interactions. There are three main arguments for such a decision: a) orchestration versus choreography; b) two-way synchronous choreography task; c) native derivation from business process choreographies.

a) orchestration versus choreography. RESTful interactions are interactions among participants that individually obey to the REST architectural style constraints (see Section 2.6). Considering the client-server constraint, if two or more participants engage into a business to business interaction, they have to take the client and server roles depending on the domain context. Usually the server role is played by service providers on the Web (e.g., `www.airbnb.com`, `open.hpi.de`, `https://easychair.org/` and `www.idealo.de`) and the client role is mostly played by the customers who visit these Web services (most often via a browser). However, there are cases where two or more service providers engage with each other to reach a common business goal, for example, *ARS* charges the *Guest* via a *Payment Organization* (Figure 19). Since REST requires a client-server relation, the client and the server roles need to be switched between participants based on who initiates the interaction. For example, *ARS* takes the role of the server for the *Host* and the *Guest* (both being clients), but it takes the role of the client when it requests the payment to the *Payment Organization* (the server). Let us call this notion *dynamic client-server role assignment*.

One solution for capturing the RESTful interactions is to consider the perspective of each service provider and capture the behavior of incoming requests (from the clients) and outgoing responses (from the server). This behavior can be modeled by orchestration models, for example, BPMN process models as in [76, 77] or UML sequence diagrams as in [28]. We would need to model the respective orchestration for each server and, in addition, the way the servers communicate with one another. For example, using BPMN process collaboration diagrams the individual behavior of the servers can be captured in process pools and their interaction via messages exchanged between the pools (see section Section 2.3).

We argue that modeling the behavior from the local view without the global view has limitations. Representational State Transfer (REST) revolves around the concept of the resource. A resource is stored and managed by a server and a client interacts with resources via resource representations. Due to the *dynamic client-server role assignment*, explained above, it becomes hard to manage the resources from the local point of view. When it comes to interactions between multiple participants, it is important to focus on a global perspective in order to capture the state of common resources and the allowed interactions with these resources at any point in the interaction. Especially when common

resources concern multiple non-human systems, the systems need to agree and coordinate on how to manage the resources at design time (for example the *reservation* or *payment* resource).

Therefore, we decided to use a choreography language for capturing RESTful interactions. After reviewing several choreography languages (listed in Section 2.4) we opted for the BPMN choreographies, despite its limitations [10].

b) two-way synchronous choreography task. Another reason is that choreography diagrams are synchronous ([99]) in that a sent message is always received and the sender waits for the receive confirmation before it continues. This fits well for modeling RESTful HTTP interactions because for every request there is a response that confirms the message being received. Moreover, the choreography task (the main composing unit of the a choreography model) can model two-way communications allowing HTTP request and response to be captured in a single choreography task.

c) native derivation from business process choreographies. Extending BPMN business process choreographies to model RESTful interactions facilitates the derivation of the latter from the former (Requirement 2). The global perspective is kept unchanged and the derived REST interfaces can directly reflect the business intent expressed by the business process choreography. We are of the firm belief that the REST interfaces should not be designed in a bottom-up fashion in such a way that they reflect the IT system behavior. Rather, the behavior of the IT system should enable the REST interfaces to support a specific real world scenario. "REST interfaces should be designed and configured, not coded"[46].

5.2 RESTFUL CHOREOGRAPHY LANGUAGE SPECIFICATION

The RESTful choreography modeling language is an enhancement of BPMN choreography modeling language with REST-specific information. To express the enhancement, we provide a metamodel extension of BPMN 2.0 choreographies, a formal mathematical specification and a graphical annotation, each of which is described in details in the following subsections respectively.

5.2.1 *Metamodel*

Business process choreographies capture the interaction between participant from the global perspective. In case of RESTful interactions we want to capture REST-specific interactions between participants. The choreography task is the smallest unit for composing a choreography model. It expresses an initiating message sent from one participant (the initiator) to another (the recipient) and, optionally, a respond message sent in the opposite direction. Based on the BPMN 2.0 choreography metamodel, every choreography task is related to one or two message flows, where each message flow is concerned with exactly one message,

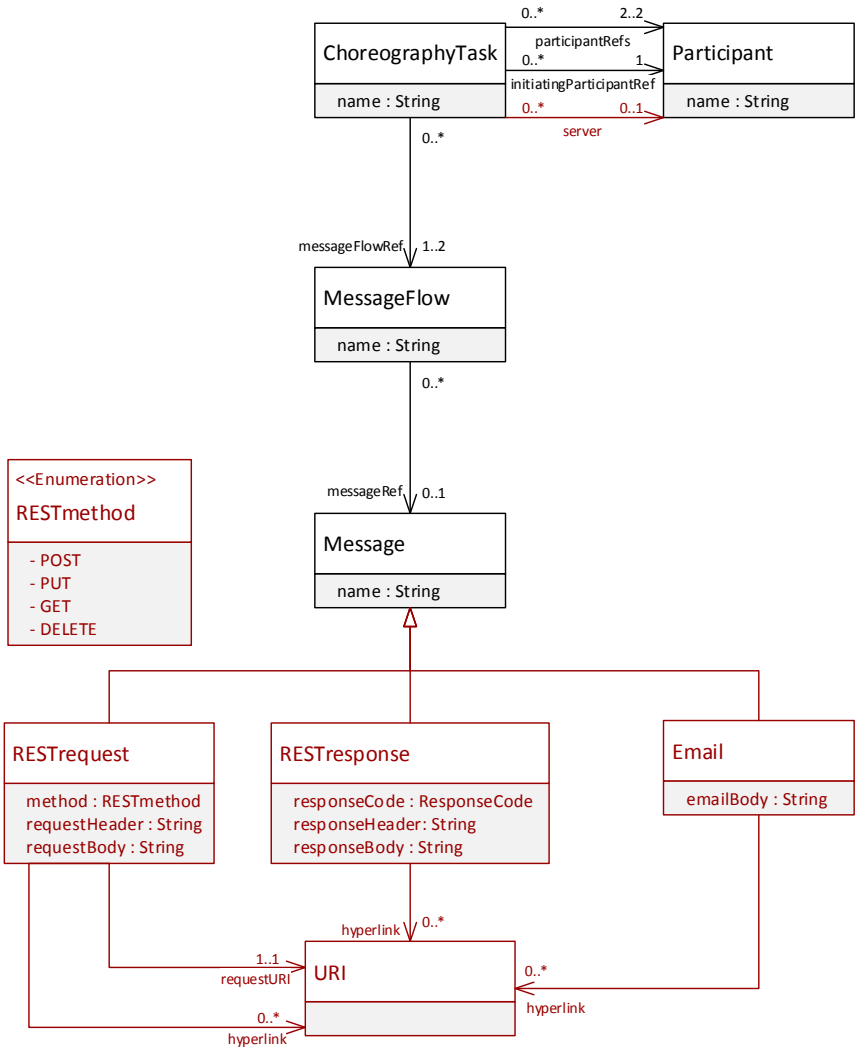

Figure 20: The extension of BPMN 2.0 choreography metamodel for modeling RESTful choreographies (new elements are drawn in red).

as shown in Figure 20. To capture the REST-specific messages we propose a specialization of the *Message* entity into three types of messages (depicted in red in Figure 20):

- *RESTrequest*. A REST request message consist of an HTTP request that must contain a REST method/verb, i.e., *POST*, *GET*, *PUT* and *DELETE*, and it must contain a request URI. In addition, it contains the header and the body of the request, where more hyperlinks can be embedded. The header and the body are optionally modeled, i.e., they can be empty strings. However, if there are embedded hyperlinks they must be specified accordingly via the *hyperlink* association to the *URI* class. The REST verb determines the method applied to the resource that is identified in the *requestURI*. We use HTTP [22], because virtually all the cases of

RESTful interactions on the Web take place over HTTP. However, REST is an architectural style and it is decoupled from the underlying protocol.

The REST verbs intrinsically are insufficient for relying domain specific information beyond their simple semantics (Table 1). For example, there exist no REST request such as *DECLINE /reservation/id* for the *Host* to sent to *ARS* in order to decline the reservation. We want to allow the participants to read and understand the progress of a given REST resource without having to access it for retrieving detailed information. To this end, we propose a specific schema for the URI, namely *domain/resource/id/state*. The participants can read the details about the resource's current state by retrieving a resource representation via GET *domain/resource/id/state*. They, then, can propose a new state by sending a new resource representation via *PUT domain/resource/id/newState*. For example, the *Host* can read details of the reservation and declining it by sending the respective REST requests to *ARS*: GET */reservation/id* and *PUT /reservation/id/declined*. The behavior of the resources is managed in a similar fashion as the behavior of data objects in a BPMN 2.0 business process model, albeit, data objects are managed only locally despite being share-able via messages.

- *RESTresponse*. A REST response message must contain an HTTP status code that informs the requester about the effect of his request on the server, e.g., 200 OK, 201 Created, 404 Not Found, 204 No Content. Depending on the answer, a URI can be embedded in the response body. Same as in the REST request case, only the hyperlinks are required to be modeled.

- *Email*. In an interaction on the Web, the recipient may not always provide a RESTful API, e.g., the *Guest* and the *Host*. The problem of notifying such kind of a participant can be solved, besides other means [77], via an email-based approach. In this case, the service provider plays the role of the initiator, whereas the customer plays the role of the recipient. As initiating message, we use an email message that might contain further links for the client to follow in the forthcoming choreography tasks. In this case, there is no need to model a return message. We use the email message type as a placeholder for all kind of information passed that is not a REST request, for example, a bank sends a post message that contains instructions including hyperlinks, which the customer can follow in order to use the online banking services. As with the previous two message types, the hyperlinks (if any) are required to be modeled.

A choreography task has exactly two participants, one of which is the initiator (see *initiatingParticipantRef* association in Figure 20). From

this information alone, the participant's client and server roles, in terms of the client-server REST constraint, are not always obvious. The roles can, however, be derived by identifying the sender and the recipient of the REST request message. Nevertheless in order to enforce consistency between the server role and the REST request, the *server* association is added from *ChoreographyTask* to *Participant*. A choreography task can have up to one participant that plays the role of the server. Every time a choreography tasks references a REST request, response or both there must be a participant that plays the role of the server. On contrary, no server role is played by the task's participants in the case of a choreography task that references an email message.

For the remainder of this section we focus on the relations among the REST request message, REST response message and email message with respect to the choreography tasks. If a choreography task refers to a REST request message, then the REST request message must always be the initiating message of the task at hand. If the same choreography task refers also to a REST response message, we call it a REST task. In sum, a REST task encompasses a full RESTful HTTP request and response. For example , the *Guest* sends to *ARS* the request *POST /reservation HTTP/1.1* and *ARS* responds with the response *HTTP/1.1 201 Created Location: /reservation/id/details*. Likewise, if a choreography task refers to an email message the email message must be the initiating message. We call this task, naturally, an email task.

In a REST task, the initiator plays the role of the client and the recipient the role of the server. However, there is a case where the REST request and response can be captured by more than a single two-way choreography task. Depending on the scenario, it might be useful to model different server responses for the same client request. For example, when the *Guest* sends a reservation request to *ARS*, ARS might have enough information about the availability calendar of the *Host* to respond whether the request will be forwarded to the *Host* or will be rejected due to unavailability. Since the server can send only one response per request, the possible responses are all mutually exclusive. In addition, the client does not know the response until it arrives. Thus, we are under the exact conditions of using a choreography event-based gateway for modeling said behavior. Figure 22 depicts this particular case. The REST request message is captured as an initiating message by a choreography task that is immediately followed by an event-based gateway. The gateway is directly followed by as many choreography tasks as the possible number of answers from the server. [1] The response tasks must each reference only a single message—an initiating message that captures the REST response from the server to the client. This is

1 Note that we model alternative responses that are relevant from the business logic point of view. We normally discard HTTP level responses that describe unsuccessful requests and rather assume the happy case of all requests being successfully addressed.

the only case where the REST request-response pairs are not modeled by a single two-way choreography task (REST task).

In sum, a REST task models a RESTful request and response between the client (initiating participant) and the server (receiving participant) as shown in Figure 21 a). In a multiple response case by the server, an event-based gateway splits the request task from the responds tasks, as depicted in Figure 22 for the case of two responses. For every non-RESTful interaction we use the email task, as shown in Figure 21 b).

5.2.2 *Formal specification*

In this section we, first, introduce a formal definition of the business process choreography, over which the RESTful choreography diagram is subsequently defined. The formalization of the business process choreography is not a complete one to one mapping of the BPMN 2.0 choreography specification [65] but is limited to only the concepts needed for our extension, e.g., we do not define the call choreography or the sub-choreography.

Definition 5.1 (Business Process Choreography)**.**
A Business Process Choreography is a tuple $C = (N, S, P, M, etype, gtype, init, recip, initm, retm)$ where:

- $N = T \cup E \cup G$ is a finite set of nodes where:
  - T is a non-empty finite set of choreography tasks.
  - E a is finite set of events.
  - G is a finite set of gateways and the sets $T, E, G \subseteq N$ are all pairwise disjoint.
- $S \subseteq N \times N$ is a set of sequence flows.
- P is a set of participants.
- M is a set of messages.
- $etype : E \to \{start, intermediate, end\}$ assigns an event type to each event.
- $gtype : G \to \{xor, ebased, or, and\}$ assigns a gateway type to each gateway.
- $init : T \to P$ assigns a participant as initiator to each choreography task.
- $recip : T \to P$ assigns a participant as recipient to each choreography task.
- $initm : T \to M$ assigns a message as initiating message to each choreography task.
- $retm : T \to M \cup nil$ assigns a message as return message to each task. nil stands for no return message.

◀

Then we have:

Definition 5.2 (RESTful choreography)**.**
RESTful choreography $C_R = (N, S, P, M, U, V, etype, gtype, init, recip, initm, retm, mtype, verb, reqURI, server, hyperlink)$ is an extension of business process choreography C with the following additional concepts:

- U is a set of URIs.
- $V = \{POST, PUT, GET, DELETE\}$ is a set of the most commonly used REST verbs.
- $mtype : M \to \{req, res, email\}$ maps any message exchanged in a RESTful choreography to one of three message types: request; response; and, email.
- $verb : M_{req} \to V$ assigns a REST verb to every rest request, where $M_{req} = \{m \in M \mid mtype(m) = req\}$ is the set of all REST request messages.
- $reqURI : M_{req} \to U$ assigns a request URI to every REST request.
- $server : T \to P \cup nil$ assigns a participant as a RESTful server to each choreography task.
- $hyperlink : M \to 2^U$ maps each message of the RESTful choreography to a set of URIs that can be used by the clients as hyperlinks in order to continue the interaction with the server.

◀

A choreography task has a (RESTful) server participant iff it models a REST request or a REST response or both. Formally, $\forall t \in T, server(t) \neq nil \Leftrightarrow (mtype(initm(t)) = req \vee mtype(initm(t)) = res)$. Note that, in an email task, an initiating participant is not assigned as a server, despite of being able to provide a RESTful API, because it does not make use of its server capabilities when sending an email. For example, *ARS* does not play the role of the server in the tasks that send email notifications to the *Guest* or the *Host*. However, it plays the roles of the server in all tasks where the *Guest* or the *Host* send a REST request.

Request messages can only be used as initiating messages. Formally, $m \in M_{req} \Rightarrow \nexists t \in T \mid retm(t) = m$. Likewise, email messages can only be used as initiating messages: $m \in M_{email} \Rightarrow \nexists t \in T \mid retm(t) = m$, where $M_{email} = \{m \in M \mid mtype(m) = email\}$. Response messages can be return messages (in case of a REST task) or initiating messages in aforementioned case of the server multiple response.

Definition 5.3 (REST task)**.**
A REST task is a task $t \in T$ such that $mtype(initm(t)) = req \wedge mtype(retm(t)) = res$. ◀

Likewise:

Definition 5.4 (email task)**.**
An email task is a task $t \in T$ such that $mtype(initm(t)) = email \wedge retm(t) = nil$. ◀

As described previously, the only way a REST request-response pair can be split from a single task is by introducing an event-based gateway between the REST request and the multiple alternative REST responses. To formally express this property we first need the following notations.

Let $p_a = (n_1, n_2, ..., n_k)$ be a path in C_R such that $\forall i = 1..k-1, (n_i, n_{i+1}) \in S$ of C_R. Then, we denote with $t\Box = \{t' \in T \mid ((t, t') \in S) \vee (\exists p_a = (t, n_1...n_k, t') \wedge \forall i = 1..k, n_i \in N \setminus T)\}$ the set of all choreography tasks that directly succeed t (gateways and events are excluded). Meanwhile, we denote $n\bullet = \{n' \in N \mid (n, n') \in S\}$ the set of all nodes that directly succeed node n. Since task $t \in T \subseteq N$, $t\bullet$ is the set of all nodes that directly succeed task t. In a similar fashion we denote $\bullet n$ and $\bullet t$ as the set of all direct preceding nodes of respectively node $n \in N$ and task $t \in T$.

At this point, we can express formally the case of the server multiple response (see Figure 22):
$\forall t \in T, (mtype(initm(t)) = req \wedge retm(t) = nil) \Rightarrow$
$|t\Box| \geqslant 2 \wedge \forall t' \in t\Box, init(t') = recip(t) \wedge recip(t') = init(t) \wedge$
$mtype(init(t')) = res \wedge retm(t') = nil \wedge$
$\exists g \in G \mid \bullet g = \{t\} \wedge g\bullet = t\Box \wedge gtype(g) = ebased.$

5.2.3 *Graphical annotation*

Graphically, RESTful choreographies are an enhancement of BPMN process choreographies with REST-specific annotations. To visually enrich the choreography diagram with REST information, we annotate the messages associated to choreography tasks. REST-annotated messages are sufficient for reliably expressing REST specific information—a proposition that will be clear in the first part of the thesis main contribution (Chapter 5, Chapter 6 and Chapter 7). The annotation of messages spares the introduction of entirely new graphical modeling elements, and hence, avoids requiring a new underlying metamodel for designing RESTful choreography diagrams. The main drive for this decision is that we want the Web engineers to be able to model and use RESTful choreographies with existing modeling tools and technologies that already support BPMN 2.0 process choreographies, e.g., Eclipse BPMN2 Modeler [2] and Signavio Process Editor [3]. Nevertheless, an instance of the RESTful choreography metamodel (introduced in subsection 5.2.1) can be generated by parsing the annotated messages and retrieving the REST-specific information, for example, an XML Metadata Interchange (XMI) file [66] can be created based on the RESTful choreography metamodel.

2 https://www.eclipse.org/bpmn2-modeler/
3 https://www.signavio.com/

A REST task is annotated with a REST request message and a REST response message like shown in Figure 21 a). The request message is composed of a concrete REST verb followed by the URI that identifies the resource on which the REST method is applied. In this thesis, we use the four most used REST verbs/methods: *GET*; *POST*; *PUT*; *DELETE*. The request message is always drawn with a clear white filling because it is always a initiating message (see Section 2.4). The response message starts with an HTTP response status code based on the HTTP/1.1 standard [29]. Both request and response messages might contain hyperlinks in their body. The response message icon is shaded because, in a REST task, the response message is always a return message. Otherwise, a response message can be an initiating message only in the case of multiple responses from the server as depicted in Figure 22.

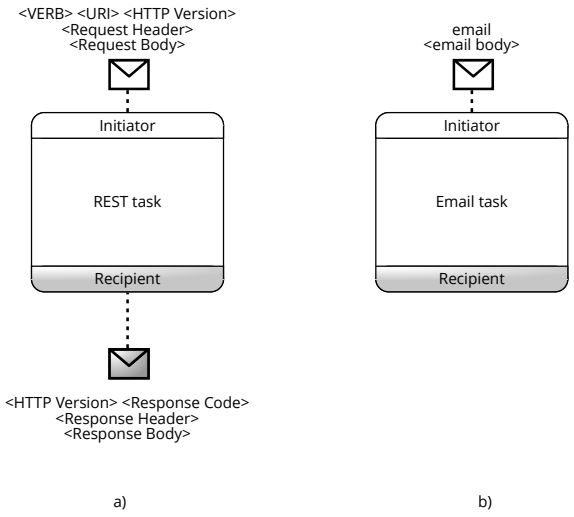

Figure 21: Enriching choreography tasks by REST-specific annotations

Figure 21 b) depicts an email task. In an email task, the email message is an initiating message and there are no return messages. If the body of the email (the one that the choreography describes or prescribes) contains hyperlinks, they have to be explicitly reflected in the choreography diagram.

The information in the HTTP requests/responses or email messages does not need to be at the same level of detail as that of the implementation, which might include, for instance, text fragments. However, it is essential to include the hyperlinks that eventually determine the behavior of the entire interaction modeled in a RESTful choreography diagram. In the ARS scenario, for example, it is essential to include links in the email messaged to the *Host* allowing him/her to accept or decline the reservation request by following those hyperlinks. Therefore when modeling a RESTful choreography diagram, all hyperlinks must be reflected in the model allowing us to capture one of the main properties of REST—hypermedia as the engine of the application state

(see Section 2.6). This makes hyperlinks very important for analyzing the behavior of a RESTful choreographies as it is shown later in Chapter 7.

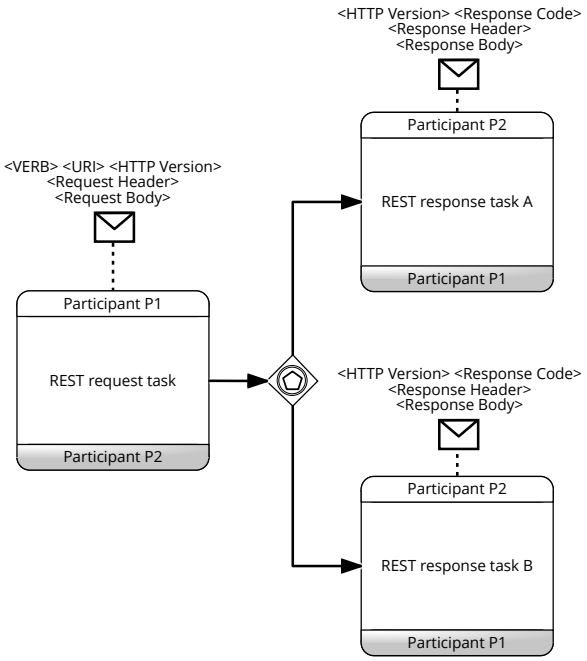

Figure 22: The modeling construct for the case of a server multiple response (two responses in this case)

In general, choreography gateways are used in a RESTful choreographies in a similar manner as in BPMN choreographies. No special enhancement is required. Nevertheless, hyperlink can be used on conditional flows that follow a choreography exclusive or inclusive gateway. Following the choreography gateways rules, the hyperlinks have to be shared among the gateway-affected participants upfront. In this thesis we tackle the delicate case of executing choreographies' exclusive gateways. Please refer to Chapter 8 for more details.

5.3 DERIVATION GUIDELINES AND DESIGN PATTERNS

Having introduced the RESTful Choreography Language, we provide some derivation guidelines and design patterns that we have encountered during the design of several RESTful choreographies spanning different application domains. For illustration, we transform the *ARS* choreography diagram from Figure 19 into the RESTful choreography depicted in Figure 24.

In the following, we introduce a step-by-step guide on how to enrich a choreography diagram with REST annotation exemplified with the *ARS* choreography:

1. *Identify the REST Servers.* Given a choreography diagram, we first have to identify the participants who play the role of the server in terms of REST. This is crucial for determining the type of RESTful interactions between the participants in the following steps.

   In the *ARS* choreography from Figure 19, the *ARS* and the *Payment Org.* are the only two participants who provide a RESTful API. Hence, the rest of the participants, namely the *Host* and the *Guest*, can only play the role of the REST client.

2. *Designate the REST and email tasks.* Knowing which participant provides a REST API (is a REST server) allows us to transform the choreography tasks into REST tasks or email tasks. If the recipient of the initiating message provides a REST API than the task is designated as a REST task. Otherwise, if the participant who receives the initiating message does not provide a REST API, we are under the conditions of an email task. The case in which the server may directly reply with alternative responses can be modeled by either a REST task that is followed by mutually exclusive email tasks (which represent respectively the different responses) or by the construct introduced above in Figure 22. The choice between the two depends on the domain context or on the preferences of the REST API designer.

   In the *ARS* choreography there are different tasks between the four participants. The *Guest* and the *Host* interact only with *ARS*. Since ARS is a REST server, they send REST requests to *ARS* and receive information either as part of the REST responses or via emails. Despite the *ARS* providing a REST API, it has to play the role of the REST client when it sends a payment request to the *Payment Org.* Hence, the task that models this interaction is a REST task as shown in Figure 24. The figure considers the case where the *ARS* confirms the reservation request and either continues the interaction (by forwarding the reservation request to the *Host*) or it sends a "Not-possible" email to the *Guest*. An alternative solution is depicted in Figure 23, where the *ARS* replies to the *Guest* via the two alternative HTTP responses: *HTTP/1.1 201 Created link: /reservation/id/details* (the request is accepted); and, *HTTP/1.1 201 Created link: /reservation/id/not-possible* (the request is not possible).

3. *Identify the main REST request URIs.* In order to identify the main request URIs (see metamodel in Figure 20), we have to identify the main REST resources. In a similar fashion that we follow to identify data objects in a process model, we need to identify resources in a RESTful choreography. Of course, the REST designer must have some domain-specific information to identify the REST resource and model its behavior. The REST resources always lie on the server side, and, hence, the servers responsible for the re-

sources have to be identified. Last, the resource states and the way they change during the choreography run have to be specified.

For example, the *ARS* choreography is revolved around the reservation of an accommodation place. That means that the *reservation* is a prime candidate for being a REST resource. Since the *ARS* is responsible for the *reservation*, the *Guest* and the *Host* can access and modify it by sending REST requests to *ARS*. The *reservation* resource goes throw different states in the choreography, i.e., created, not-possible, accepted, confirmed, declined, and expired. As for the *Payment Org.*, it is only concerned about the *payment*. Thus, the *payment* is a REST resource that is managed by this particular server.

4. *Complete the hyperlinks* The information determined in the previous steps, are sufficient for specializing all the messages to accordingly REST request, REST response and email messages as specified in the RESTful Choreography Language. Last but not least, the designer of the RESTful choreography has to make sure that all the REST clients are, at any point in the choreography, in possession of the hyperlinks that will allow them to successfully continue the interaction with the servers (see HATEOS principle in Section 2.4).

   In Figure 24, the *Host* is provided with three hyperlinks for checking the requested reservation in details and for directly stating his choice on the reservation.

We have observed several patterns that reoccur in RESTful choreographies concerning the relation between the content of the messages exchanged and the ordering of choreography tasks. Since the hyperlinks sent between participants pave the way for upcoming interactions we focus on how the number of hyperlinks referred in the task impacts the consecutive tasks of the RESTful choreography. The presented designed patterns can be taken into consideration when modeling a RESTful choreography, but they should not be necessarily enforced because they make no claim on the correctness of the model. We introduce formal properties later in the thesis to enforce certain relation between hyperlinks and tasks that must hold in order for a RESTful choreography to be correct (see Chapter 7).

- *no-hyperlink pattern.* In case of the no-hyperlink pattern, the choreography task incorporates a REST response message or an email message without any hyperlink. This can be a simple notification, e.g., the information about a resource being deleted. This choreography task is usually followed by an end event or by a choreography task that does not involve the original recipient. In any case, the missing hyperlink hints to a lack of future conversation between these two particular participants. Theoretically,

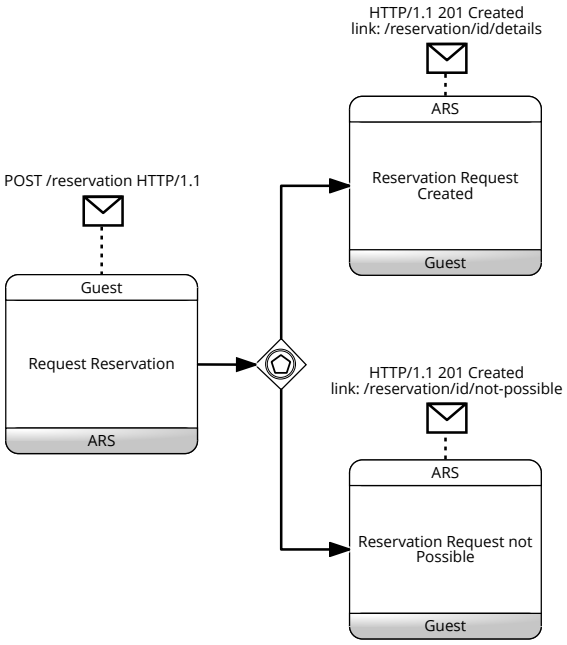

Figure 23: An alternative solution for modeling the way the *Guest* is informed about the reservation

however, the recipient might initiate the direct succeeding REST task by using a hyperlink that it received earlier in the choreography. However, we have not encounter such a case during our practice of modeling RESTful choreographies.

- *multi-hyperlink pattern.* In case of the multi-hyperlink pattern, the choreography task incorporates a HTTP response message or an email message with multiple hyperlinks. This task usually represents a set of choices (not necessarily mutually exclusive) given to the recipient. In that case, the task is followed by an event-based gateway or parallel gateway that represents the possible future interactions (alternative or parallel) the recipient can follow through the hyperlinks. If the hyperlinks refer to the same REST resource with different states then the hyperlinks lead to mutually exclusive requests. Otherwise, it depends on the context and the domain whether there is a upcoming parallel split or not. Note that such behavior can be captured also with inclusive gateways but we try to avoid them due to their complex semantics [8, 19, 96].

- *single-hyperlink pattern.* In the single-hyperlink pattern, the choreography task incorporates a REST response message or an email message with a single hyperlink. This kind of message, generally, is a notification, which can link to additional information to what is originally included in the message body. Typical examples are a HTTP response informing about the effect of the REST request on the resource, or an email linking to a new resource or the state

of a recently changed one. This kind of REST task is the most commonly encountered task in RESTful choreographies and not much can be inferred about the follow-up tasks.

Applying the derivation design guidelines and patterns to the *ARS* choreography yields the RESTful choreography in Figure 24. The interaction starts by the *Guest*, who creates the resource *reservation* via a *POST* verb to the *ARS*. Presuming that the *ARS* knows some details about the availability of the *Host*, it can either send back an email stating the unavailability of the *Host* or it can forward the reservation request to the *Host* via email. In the case of the latter, the *Host* is provided with three links: a link to which the *Host* can send a *GET /reservation/id/details* request for further reservation details; two links which allow the *Host* to respectively decline or accept the reservation request.

In case of acceptance, the *Host* sends a *PUT* request to change the state of the *reservation* resource to *accepted*. The *ARS*, then, sends a payment request to the Payment Org. (here it is assumed that the *Host* payment address is already known by the *ARS*). If the payment is successful, the *ARS* informs the *Host* and the *Guest* via an email, which it embeds the hyperlink */reservation/id/confirmed* for retrieving the new state of the *reservation* via a *PUT* verb.

Contrarily, the *Host* sends a *PUT* request for changing the state of *reservation* resource to *declined*. The *ARS*, then, informs the *Guest* via an email like in the previous case. The email contains an hyperlink for the *Guest* to send a *GET* request for retrieving further details that, perhaps, describe the reason for the declination.

In case 24 hours have passed, the *ARS* changes the state of the *reservation* resource internally to *expired* and sends the corresponding link */reservation/id/expired* to the *Guest* and the *Host* via emails. They can, at any time, send a *PUT* request with the link to obtain details around expired reservation request.

5.4 CONCLUSION

We showed how a business process choreography model can be enriched step-wise with REST-specific annotation to finally attain a RESTful choreography. However, there is a conceptual gab between the business process choreography diagrams and RESTful interactions. Choreography diagrams are normally modeled from business domain experts while REST interfaces are designed by Web engineers. Deciding on which REST resources are important, the right hyperlink structure, REST versus email tasks, and refinement at the RESTful interaction level still require manual work and REST expertise from the RESTful choreography designer. In order to mitigate these problems and facilitate the work of the RESTful choreography designer, we provide an approach that aids and expedites the creation of RESTful choreographies by reducing greatly the manual work. The approach, presented

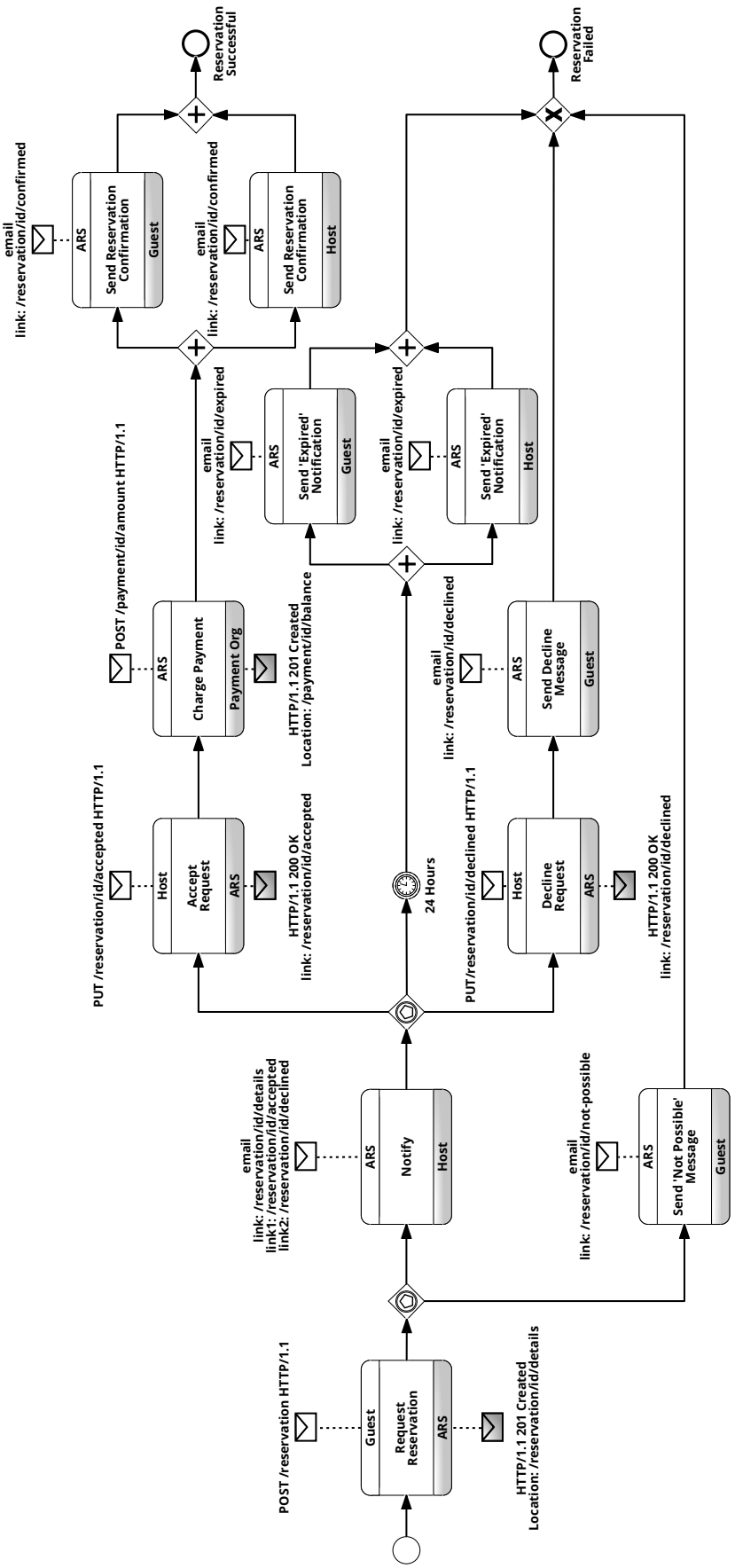

Figure 24: RESTful choreography diagram of the motivating example

in the next chapter, provides a semi-automatic method that employs natural language processing techniques for extracting domain specific information from existing business process choreographies.

Whether the RESTful choreography is manually or semi-automatically derived, we still have no completeness criteria with respect to REST, e.g., do the introduced hyperlinks induce deadlocks or are the REST resources properly accessed and modified? To answer these questions we introduce in Chapter 7 formal completeness properties that guarantee the lack of REST-induced deadlocks in a RESTful choreography.

6

SEMI-AUTOMATIC DERIVATION OF RESTFUL CHOREOGRAPHIES

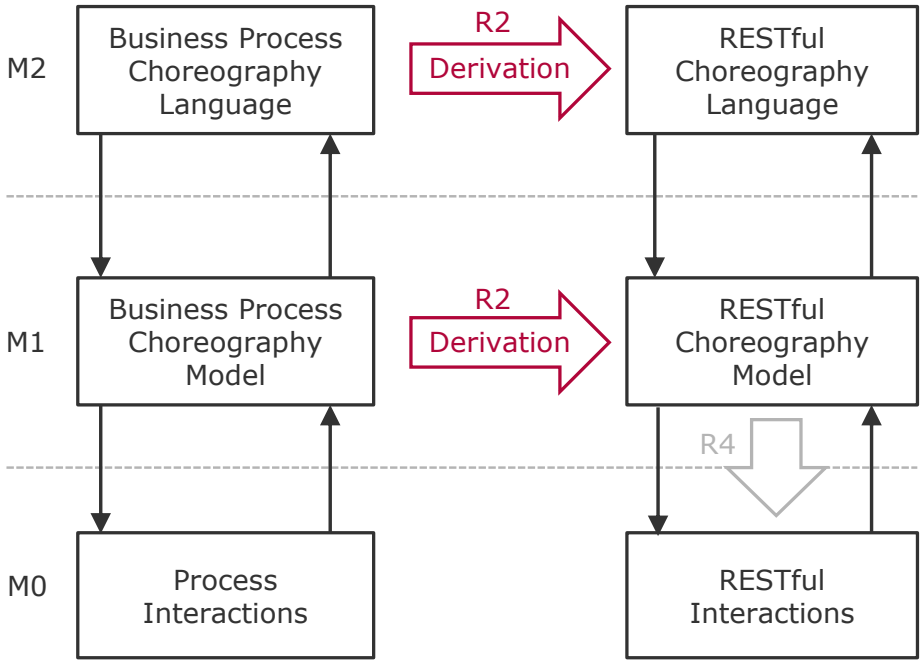

Figure 25: Semi-automatic generation of RESTful choreographies

In the previous chapter we introduced RESTful choreography language for capturing RESTful interactions among Web services and their customers into models. The language can be used prescriptively or descriptively. In addition, we provided guidelines for the top-down approach of arriving from a business process choreography to a RESTful choreography. However, this includes manual work and requires expertise in both business process management (BPM) and RESTful API design. Ideally, we would like to have a clear separation of concerns between business process choreography modelers and Web engineers.

In this chapter we address requirement R2 for automating the derivation of RESTful choreography diagram from business process diagrams. One key challenge is the identification of the REST resources and the REST methods that access and manage these resources. Usually this requires knowledge about the contextual domain where the choreography is situated. In a business process choreography model, the contextual domain knowledge is expressed in the labeling of the choreography tasks. For example, the choreography task with the label *Accept Reservation* hints at the modification of *reservation*, which can be seen as a REST resource, to the state *accepted*, which can be implemented via

the REST verb PUT. If this particular choreography task needs to be implemented as a REST task on the Web, the request and the response messages would be specified as respectively *PUT /reservation/id/accepted HTTP/1.1* and *HTTP/1.1 200 OK hyperlink: /reservation/id/accepted*.

We employ natural language processing (NLP) techniques to generate the REST-specific information from the business contextual information in process choreography tasks. The proposed method is evaluated in terms of effectiveness resulting in the intervention of Web engineers in only about 10% of the all generated REST tasks. The comprehensive evaluation is presented in Chapter 10. This chapter is based for the most part on our published papers [58] and [60].

6.1 PROBLEM STATEMENT

The business process choreography diagram introduced in BPMN 2.0 [65] is a modeling language that focuses on the specification of the interactions between two or more participants, who, in general, are business actors, e.g., enterprises, customers, or organizations. Compared to business process models, the choreography diagram abstracts from the participants' internal processes and specifies the order in which the messages are exchanged between the participants.

Figure 26 depicts an example of a choreography diagram. This diagram describes the interaction between different participants involved in the submission, review, and organization processes regarding the arrangement of a scientific conference. Some of the main stakeholders in a conference include the organizers, authors, and reviewers. The choreography diagram depicts the interactions between these three participants starting from issuing a call for papers (CFP) and ending, in the best case, with the confirmation of the paper publication. To facilitate these interactions, the participants make use of a *review management system (RMS)* that, in our case, is inspired by `http://easychair.org`. The *RMS* is responsible for coordinating these three participants throughout the entire collaboration.

As explained in Section 2.4, the main composing element of a choreography diagram is the choreography task, which is our main object of concern for this chapter. It represents message exchanges between the initiator and the recipient. The return message is optional and can be sent from the recipient to the initiator. For example, the choreography task *Create CFP* has only the initiating message, while the choreography task *Submit Paper* has also a return message as a confirmation that the paper submission has been successfully received. The messages are not depicted visually in the diagram because they are not relevant to the approach introduced in this chapter.

The choreography diagram in Figure 26 depicts the order of the interactions in which the *Organizer*, *RMS*, *Author*, and *Reviewer* engage in order to reach the final goals (see the end events): *Paper rejected*, *Short*

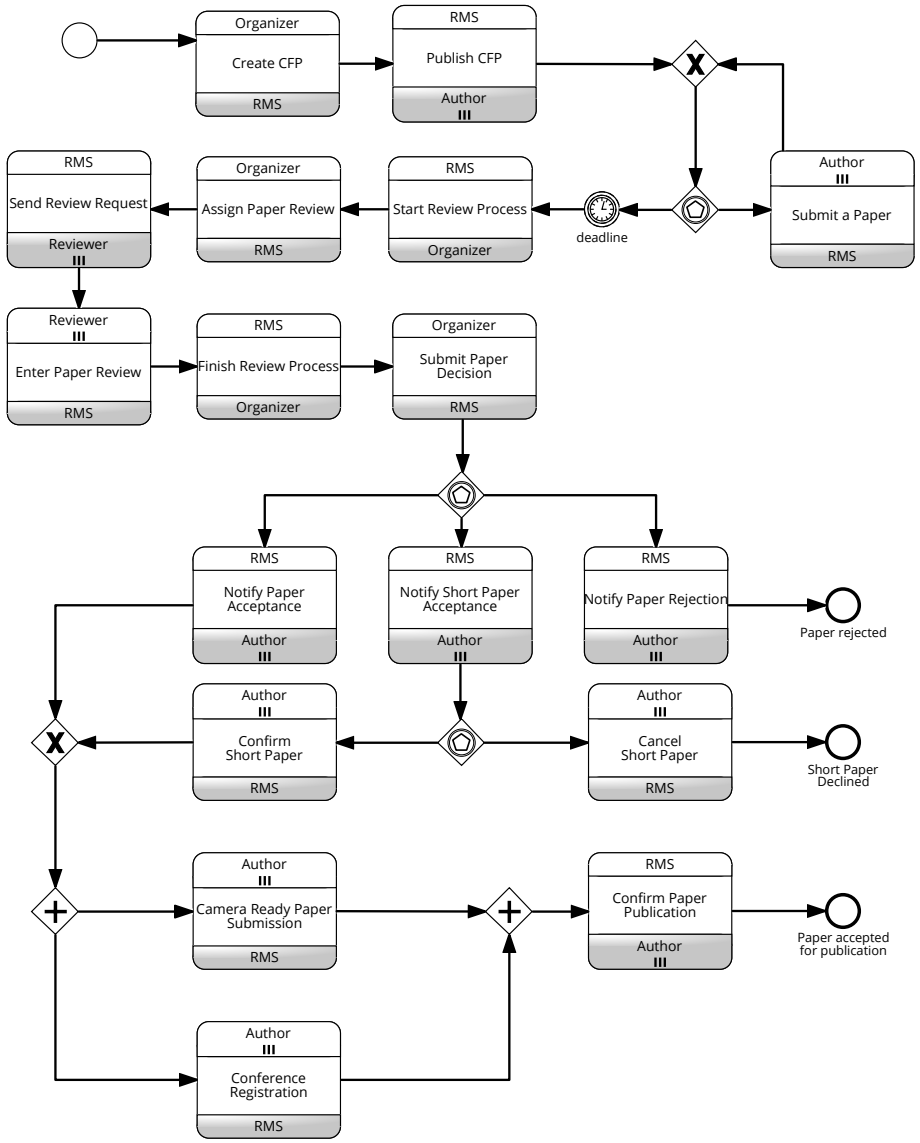

Figure 26: Choreography diagram for paper submission and review management

paper declined, and, *Paper accepted for publication*. The interaction start with the *Organizer* creating a CFP on the *RMS*. The *RMS* publish the CFP in a way that prospective authors are made aware of. This marks the beginning of the paper submission period until an absolute deadline (e.g. May 24th) has been reached. The choreography construct that expresses such behavior is an event-base gateway that expects paper submissions from the *Authors* until the intermediate timer event has been signaled by the arrival of the deadline [1].

After the deadline has been reached, the *RMS* signals the *Organizer* to start the reviewing process. The reviewing process encompasses

1 Please refer to the BPMN 2.0 specification for the peculiar assumptions of the choreography timer events

the following sequence of interactions: the *Organizer* assigns the paper reviews to the *Reviewers* through *RMS*; upon receiving the review requests, the *Reviewers* enter their reviews in the *RMS*; the *RMS* notifies the *Organizer* about the end of the reviewing process.

Once all reviews are collected the *Organizer* makes a decision on which papers to reject, accept as short papers or accept as full papers. The decision activity is internal to the *Organizer* and is not reflected in the choreography diagram per se. However, the results are sent to the *Authors* respectively. This is expressed via an event-based gateway because the authors are not involved in the decision and are only waiting for the results. In case of a paper rejection, the choreography reaches the end event. In case of short paper acceptance, the *Author* has to make a choice on whether to confirm the short paper or cancel it. In case of the latter, the choreography reaches the next end event. In the case a full paper acceptance or short paper confirmation, the *Authors* have to submit the camera ready version of the paper and register to the conference. Only once the two previous tasks are executed can the paper publication be confirmed by the *ARS* leading to the final end event.

Given the choreography example above, the goal is to derive a RESTful choreography diagram. Let's assume that the *RMS* provides a RESTful API. This means that the *ARS* plays the role of the server (see subsection 5.2.1) and the rest of the participants play the role of the client. In order to generate the REST interfaces of the *RMS*, a Web engineer has to make sense of the domain-specific information because the REST interfaces have to be clearly understood by all prospective REST clients. For example a REST request like *PUT paper/id/submitted* shows clearly the intent of the request, which is about submitting a specific paper. This raises the question: where in a business process choreography is the domain-specific information incorporated?

As can be observed in this running example, the contextual domain-specific information in the choreography model is mediated, for the most part, by the choreography tasks labels. From the choreography labels we can tell that this choreography model is about the paper submission process of a certain conference. We can understand that papers are submitted, reviewed and then accepted or rejected. In this particular example, we want to exploit each label in every task that has the *RMS* as a recipient in order to generate the REST tasks that represent the requests and responses sent between the client and the *RMS*. This leads us to the next important question: how can we harvest the task labels in an automatic fashion for generating the REST tasks?

6.2 PRELIMINARIES

This chapter explains how to semi-automatically derive REST tasks from business process choreography tasks whose recipient provide a

RESTful API. The remaining choreography tasks, where this is not the case, are mapped automatically to email tasks (see Definition 5.4).

In order to process the textual information of the labels, it is necessary to access the information in a structured way. As a starting point, we observed that choreography tasks are similarly labeled as process activities, namely to the corresponding send task in a business process model. In Figure 7, for example, the choreography tasks *Request Payment* and *Send Payment* (on the left) are labeled exactly as their respective send tasks (on the right) despite belonging to opposite participants. We can assume that the labeling of the choreography tasks does not exhibit a global bias towards a particular participant but rather it manifests a local bias towards its initiating participant.

The two-way task case This also means that the choreography task label provides next to no information about the return message (if any). This is the main reason why we map also a two-way task to a single REST task. As pointed out in section Section 2.4, a choreography task can represent either a single message exchange or two message exchanges (i.e. a send message and a reply message). In both cases, we choose to map the choreography task to a single REST task. The initiator (client) makes a REST request to the recipient (server). Since the optional second message, according to the BPMN specification [65], is a return message, we do not consider it as a new REST request but rather as a response from the the server, which can be embedded in the REST response body (see Figure 21). However, in the special case where both task participants are (REST) servers and there is the need to explicitly map each message exchange to a REST task, the original two-way task should be decomposed to a sequence of two one-way tasks before our method is applied.

The similarity (often equality) between the labels of the choreography task and the (process model) send task allows us to repurpose a set of existing approaches and tools, developed originally in the context of business process models, for analyzing process choreographies. In particular we are interested in approaches that employ natural language analysis. In our case, we are specifically interested in the approach by Mendling et. al. in [50]. The authors show that the label of a business process activity in general, including a send task [2], contains the following components: an action and a business object on which the action is applied. For our purpose, we can safely assume that a choreography task contains the same components. For example, the label *Submit paper decision*, from Figure 26, contains the action *to submit* and the business object *paper decision*.

The action and business object are not always grammatically presented as a verb followed by a noun. They can be communicated in different grammatical variations. For example, the label *camera ready paper submission* communicates the action in a different grammatical

2 A send task is an instance of an activity in a business process model

structure by using the noun *submission*, which, in this case, expresses the same action *to submit*. To ensure the independence from the grammatical labeling structures, we rely on the label annotation approach of Leopold et al. [39] which identifies actions and business objects with a decent degree of accuracy (avg. precision: 91%, avg. recall: 90.6%).

The notions used throughout the rest of this chapter are introduced formally in the following. Let $C = (N, S, P, M, etype, gtype, init, recip, initm, retm)$ be a choreography diagram according to Definition 5.1 and L a set of set the all natural language text labels used in the choreography C. We denote with $label : T \to L$ the function that assigns a text label to a choreography task. Considering $l = label(t) \in L$ to be the label of an arbitrary choreography task t and considering W_V and W_N to describe the set of all verbs and nouns respectively, we refer to the action and the business object of l as follows:

- $\alpha : L \to W_V$ is a function that assigns an action to a choreography task label
- $\beta : L \to W_N$ is a function that assigns a business object to a choreography task label

As an example, consider the choreography task that is labeled *Submit paper decision* from Figure 26. According to the prior conceptualization, the action is given by α(*Submit paper decision*) = *to submit* and the business object is given by β(*Submit paper decision*) = *paper decision*.

In the two succeeding sections we show how we make use of the choreography tasks' action and bushiness object to generate the REST tasks of a RESTful choreography. Section 6.3 describes the core concept for generating the REST tasks in a semi-automatic fashion while Section 6.4 reuses this concept, in addition to new ones, in a more elaborate way to produce better results.

6.3 CORE DERIVATION OF REST TASKS

Based on Definition 5.3 a REST task represents a REST request as an initiating message and a REST response as a return message. The former is composed of a REST verb and a request URI, while the latter is composed of a response code and an optional hyperlink. In sum in order to derive a REST task, we need to: derive the REST verb; generate the request URI; and, insert the response code and optionally generate the hyperlink.

Figure 27 depicts the overview of core approach. We start by extracting the labels of the choreography tasks whose recipient provides (descriptive model), or wants to provide (prescriptive model), a REST interface. Then, the NLP approach introduced above is applied to the extracted label to retrieve the action and the business object components.

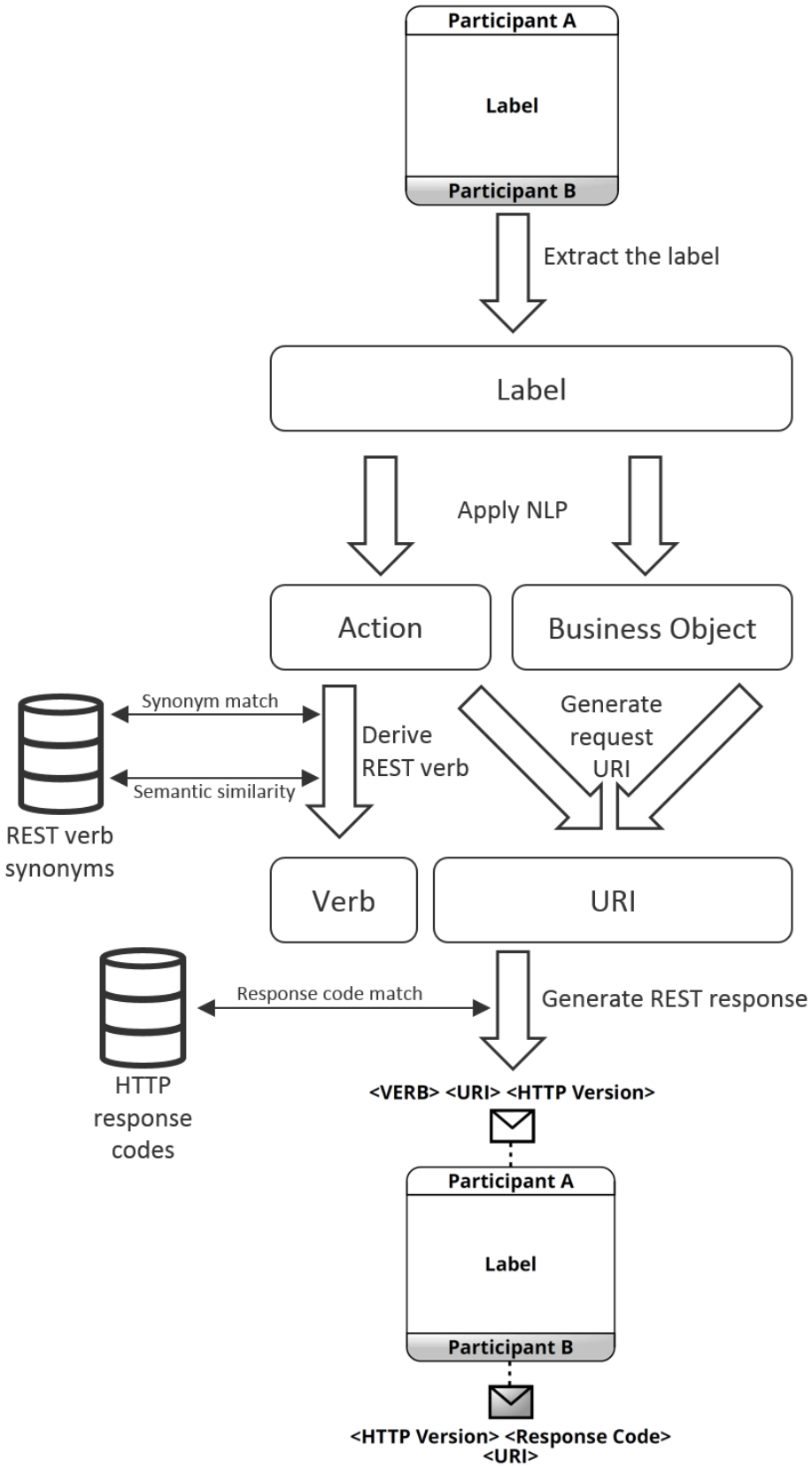

Figure 27: Overview of the core approach for deriving REST tasks

The action is used to derive the REST verb. This is achieved by finding the REST verb that is the closest to the action in terms of semantics (subsection 6.3.1). Once the appropriate REST verb is selected, it is used together with the business object to generate the request URI. The business object determines the REST resource and the action determines the requested resource state change. Together they form the request URI (subsection 6.3.2). Last, the REST task is completed by generating the REST response, which includes the insertion of the respective HTTP response for the given REST verb (subsection 6.3.3). These derived components suffice for the assemble the REST task.

6.3.1 *Derivation of the REST verb*

The general idea of deriving REST requests via natural language analysis is based on the observation that the REST resource represents the label's business object, e.g., *CFP*, *paper*, *short paper*, *conference*, *reservation*, *payment*. Consecutively, the REST verb/method represents the action taken upon the business object, e.g., *create*, *submit*, *confirm*, *register*, *send*. Therefore, we want to find the REST verb that conveys the same meaning as the label's action, or, in case we do not find an identical meaning, we want to find the REST verb that has the closest meaning to the action.

The REST verb derivation applies two steps. The first step compares the action of the respective choreography task label with a set of synonym words pertaining to a certain REST verb. More precisely, we look at the inclusion relation. This is the case where a REST verb has a synonym relation to the action. The second step involves a linguistic similarity analysis of the label's action and the synonym words, in case the action of the label does not exactly match with any of the synonym words. In the following, we discuss these two steps in further detail.

POPULATING THE SET OF REST VERB SYNONYMS. First, we require a set of words that are synonyms to a given REST verb before we can conduct the derivation. We have four sets in total as shown in Table 2. We denote this sets accordingly: $\mathrm{Syn}_{\mathrm{POST}}$; $\mathrm{Syn}_{\mathrm{PUT}}$; $\mathrm{Syn}_{\mathrm{GET}}$; and, $\mathrm{Syn}_{\mathrm{DELETE}}$. The general case is denoted with $\mathrm{Syn}_v, v \in V$, where V is the set of all REST verbs as defined in Definition 5.2. The challenge of populating these sets consists in that the REST verbs are semantically associated with a specific technical meaning that does not necessarily correspond with the original linguistic meaning of the verb. For example, the REST verb *POST* instructs the server to create a new distinguishable resource, while the verb *to post* typically describes the act of publicizing news on bulletin boards. Therefore, it is necessary to define a set of synonym words that reflect the meaning of *POST* in a technical sense. For this purpose, we asked REST experts for natural language verbs that best resemble the meaning of the REST verbs. The result of this process are the elements (not marked as derived) of the respective

Table 2: Synonym Word Sets of the REST Verbs

Verb	Description	Synonym Word Sets
POST	creation of a new resource on the server	Syn_{POST} = {create, request, $\underbrace{\text{produce, make, ...}}_{derived}$}
PUT	editing an existing resource	Syn_{PUT} = {confirm, edit, accept, send, $\underbrace{\text{support, redact, ...}}_{derived}$}
GET	retrieving an existing resource from the server	Syn_{GET} = {retrieve, read, $\underbrace{\text{get, find, recover, ...}}_{derived}$}
DELETE	deleting an existing resource	Syn_{DELETE} = {cancel, delete, $\underbrace{\text{erase, postpone, ...}}_{derived}$}

sets in the third column of Table 2. For example, the experts agreed that the meaning of *POST* is best reflected by the verbs *to create* or *to request*. As the identified verbs might not capture all the variation in language, we further consider additional synonyms that may be extracted from computational lexicons, such as WordNet [53]. For example, a *POST* verb might also be related to the verbs *to produce* or *to make*. As result, the synonym sets for each REST verb are populated with two kinds of verbs: verbs that are entered by the REST experts; and, derived verbs from computational lexicons that reflect the technical meaning of the REST verb (marked as *derived* in Table 2).

SYNONYM ANALYSIS. The *synonym analysis step* investigates whether or not the action of a choreography task label equals one of the synonym words of a certain REST verb. If this condition evaluates to true, we have found the adequate REST verb for the REST request. Otherwise, no REST verb has a synonym relation with the action. As an example, consider the choreography tasks *Create CFP* and *Confirm Short Paper*. The first task would map to *POST* because its action *to create* is a member of the set Syn_{POST}. The second task would map to *PUT* since its action *to confirm* is a member of the set Syn_{PUT}. This logic is expressed by the following function.

$$syn(l) = \begin{cases} \text{POST} & \text{, if } \alpha(l) \in Syn_{POST} \\ \text{PUT} & \text{, if } \alpha(l) \in Syn_{PUT} \\ \text{GET} & \text{, if } \alpha(l) \in Syn_{GET} \\ \text{DELETE} & \text{, if } \alpha(l) \in Syn_{DELETE} \\ \emptyset & \text{, otherwise} \end{cases} \quad (1)$$

The function requires the synonym sets to be mutually exclusive. This is indeed the case—verbs that initially belong to more than a single set, in particular those that are derived automatically from computational lexicons, are disambiguated by the REST experts.

SIMILARITY ANALYSIS. The *similarity analysis step* serves as a fallback strategy in case the *synonym analysis step* fails to assign a REST verb to a choreography task. In this case, it is necessary to find a REST verb that is most closely related, in terms of semantics, to the action. Therefore, it is necessary to determine the relatedness of an action with the synonym words. In our approach, we use the notion of semantic similarity (see e.g. [42, 83, 101]) to quantify this relatedness. We utilize the distributional similarity of the DISCO word similarity tool [34], denoted with sim_{DISCO} because it outperforms existing similarity measures [35]. Given a choreography task label l, its action $\alpha(l)$, and the set of synonym words Syn_v of a REST verb $v \in V$, the relatedness of an action of a choreography task label and a synonym REST verb set Syn_v is given as follows:

$$rel(\alpha(l), Syn_v) = \max_{w \in Syn_v} sim_{DISCO}(\alpha(l), w) \tag{2}$$

As an example, we consider the choreography task *Enter paper review* from Figure 26. Since the action *to enter* is not a member of the synonym sets of the REST verbs, we determine its relatedness to each synonym set. Applying the formula above, we receive the following relatedness values: $rel(\text{enter}, Syn_{POST}) = 0.48$; $rel(\text{enter}, Syn_{PUT})) = 0.92$; $rel(\text{enter}, Syn_{GET}) = 0.92$; and, $rel(\text{enter}, Syn_{DELETE}) = 0.55$.

Finally, we consider all of the relatedness scores to derive the most suitable REST verb for a given choreography task label. In this case, we assume that the highest relatedness score reflects the most suitable REST verb for a given choreography task. Accordingly, we assign this REST verb to the highest relatedness score. However, it might be the case that several relatedness scores are equal which consequently leads to more than one assignment of a REST verb emphasizing the necessity of a user to choose the correct REST verb. Formally, we describe the similarity analysis step as follows:

$$sim\,(l) = \{v \in V \mid \max_{v}(rel(\alpha(l), Syn_v))\} \tag{3}$$

As an example, consider again the choreography task label *Enter paper review* and its relatedness scores. Since $rel(\text{enter}, Syn_{PUT}) = rel(\text{enter}, Syn_{GET}) =$

0.92, the similarity analysis step assigns both REST verbs *PUT* and *GET* to the choreography task: $sim(l) = \{PUT, GET\}$.

The following section will explain how the request URIs are generated for the choreography tasks using the identified REST verb.

6.3.2 *Generation of the request URI*

The task of generating REST requests involves the generation of a unique resource identifier (URI) explaining how the resource is addressed via the HTTP protocol. In order to generate the request URI, we consider its generation as a language generation problem that uses the available information of the choreography task and the REST verb derivation from the previous step. Many language generation systems take a three-step pipeline approach that first determines the required information of a sentence, second plans the expression of this information, and third transforms them into correct sentences [82]. In contrast to these systems, we do not require a fully flexible approach, since the final links follow regular structures [57]. Therefore, we use a template-based approach[18, 37, 40] to generate REST URIs. In particular, we use the choreography task label together with the derived REST verb from the previous step for selecting the respective URI template. Afterwards, we fill the template with the concrete information, i.e. the action and business object of a choreography task label. It has to be noted that this approach requires the correctly derived REST verb. A wrongly derived REST verb leads to an incorrect URI prompting the REST experts to intervene by selecting the right URI pertaining to the correct REST verb.

Table 3: URI Templates for REST Requests

Request URI Template	**Example**
POST /<β(l)> HTTP/1.1	POST /CFP HTTP/1.1
PUT /<β(l)>/id/<Past Participle of α(l)> HTTP/1.1	PUT /paper/id/ submitted HTTP/1.1
GET /<β(l)>/id HTTP/1.1	GET /paperReview/id HTTP/1.1
DELETE /<β(l)>/id HTTP/1.1	DELETE /shortPaper/id HTTP/1.1

Table 3 shows the link templates for the different REST verbs and gives examples created from the choreography tasks of Figure 26. The templates emphasize that the business object of a choreography task

label ($\beta(l)$) plays an important role for the REST request URI since it resembles the server's resource that needs to be addressed by a REST verb. We therefore map the business object to the REST resource and a unique resource identifier, if the latter is known by the client. Only in case of *POST* the identifier is unknown to the client. For the remaining REST verbs the identifier is always present in the request URI. We need the identifier for accessing a specific resource via *GET* or removing the resource via *DELETE*.

In case the state of a specific resource has to be changed, the request URI should also describes the desired state change. This change is expressed by using the REST verb *PUT* and generating the past participle of the label's action. For example, the label *Submit a Paper* is translated to *PUT /paper/id/submitted* by identifying the label's business object *paper* as the main REST resource and requesting its state to change to *submitted*, which is the label's action past participle. The main reason we follow this particular template consists in designing REST APIs that convey the request intent clearly to the clients. For more details behind this design decision please refer to the introduction of the *RESTrequest* concept in subsection 5.2.1.

6.3.3 *Generation of the REST response*

The REST verb and URI constitute the REST request. With the generation of the REST response the REST task is considered complete. According to the RESTful choreography metamodel a REST response is composed of an HTTP response code and an optional hyperlink. The HTTP response code is a standardized code [29] that is returned as a part of the HTTP response for describing meta information about the server's reaction to the client's request, e.g., the request was successful, the resource is created, the resource is not found. In our approach we consider every REST task to be successful in terms of technicalities. Therefore we always assume the positive case where every request is processed accordingly by the server. As for the alternative server responses that are relevant from the business logic point of view, they are modeled as successful interactions from the REST perspective but hold different payloads that capture the respective business information (see Figure 23). Therefore, we consider only a subset of the HTTP response codes as shown in the table below.

In case of a *PUT* or a *GET* request the response code is *HTTP/1.1 200 OK*. The optional hyperlink stating the location of the resource is usually the same as the requested URI to show the client the updated resource (after a *PUT* request) or the location of the same resource (after a *GET* request). In case of a *DELETE* request, the response code is *HTTP/1.1 204 No Content* and there is no location hyperlink since the resource is permanently removed from the server. In case of a *POST* request, the response code is *HTTP/1.1 201 Created*. The location hyperlink of the *POST* request is generated similarly to the request of PUT. That

Table 4: REST response generation

REST Verb	**REST Response**	**REST Request and Response Example**
POST	HTTP/1.1 201 Created Location: <$\beta(l)$>/id/ <Past Participle of $\alpha(l)$>	POST /CFP HTTP/1.1 HTTP/1.1 201 Created Location: /CFP/id/created
PUT	HTTP/1.1 200 OK Hyperlink: <Request URI>	PUT /paper/id/submitted HTTP/1.1 HTTP/1.1 200 OK Hyperlink: /paper/id/submitted
GET	HTTP/1.1 200 OK Hyperlink: <Request URI>	GET /paperReview/id HTTP/1.1 HTTP/1.1 200 OK Hyperlink: /paperReview/id
DELETE	HTTP/1.1 204 No Content	DELETE /shortPaper/id HTTP/1.1 HTTP/1.1 204 No Content

is, the server creates the resource, assigns a resource identifier and an initial state. Hence we apply the same URI generation as the one used for generating the request URI for *PUT* (see Table 3). The difference is that in a *POST* request the resource identifier and state is determined by the server, while in a *PUT* request the resource identifier and state are determined by the client. Therefore, the hyperlink (containing the resource identifier) is requested in the *PUT* case and returned in the *POST* case.

It is important to note that the return hyperlinks are optional. They depend on the domain. For example, the reviewer can send a *GET* request and receive several hyperlinks which redirect the reviewer to each paper he or she needs to review. The exception is the *Location* field, which is a HTTP/1.1 standard response field that is used in case of a redirected resource (*HTTP/1.1 301 Moved Permanently*) or a newly created one (*HTTP/1.1 201 Created*). Thus, we use it in the response after a *POST* request, but it can also be used as a response of a *PUT*

request when the request is used to create a new resource. Since we generate a default response code and optional hyperlinks for each REST request it is up to the Web engineer to make use of them or change them according to the domain requirements. In Chapter 10, we evaluate only the generation of the REST requests because its composing elements, i.e. REST verb and request URI, are mandatory, and the generation of the response code depends solely on the derived REST verb.

Applying the core derivation approach to all the choreography tasks whose recipient provide a REST API gives us all the REST tasks of the output RESTful choreography. This means that we are left with a set of tasks that are not REST tasks. By definition of the RESTful choreography language these tasks are automatically designated as email tasks because we use email tasks as placeholders for every non RESTful interaction. Email tasks are important because they contain hyperlinks which participants can follow in order to perform RESTful interactions. The hyperlinks have to be manually entered by Web engineers in order to satisfy the HATEOAS principle (see Section 2.6) of RESTful interactions, i.e., the hyperlinks provided to the clients pave the way for future interactions with the servers. In the next chapter, we introduce formal properties to automatically check whether a RESTful choreography is enforceable in the presence of hyperlinks, i.e., the lack of an hyperlink disallows the continuation of the choreography execution.

There is, however, an exception, in that a RESTful choreography is not only composed of REST and email tasks. In Chapter 5 we specify the special case of a *server multiple response*. This is the case where a REST request is followed by multiple exclusive REST responses like shown in Figure 22, for the general case, and in Figure 23, for a concrete example. Alternatively this case can be modeled by a REST task that is followed by exclusive email tasks. The email tasks are initiated by the server to inform the client about the alternative resource state changes. It is ultimately up to the Web engineer to make a choice between the two representations as it do not change the business logic of the RESTful choreography but it is rather a technical choice.

The core derivation approach presented in this section is first introduced and evaluated in [58]. We developed a tool and applied it to choreography diagrams from different domains. Three REST experts assessed the output of the tool coming to the conclusion that the derivation of the REST verb is correct in 74.93% of the cases and the generation of the request URI is correct in 60.74% of the cases. In the next section we enrich the core approach with few additional new concepts to achieve better results. An elaborated evaluation of the advanced derivation approach is provided in Chapter 10.

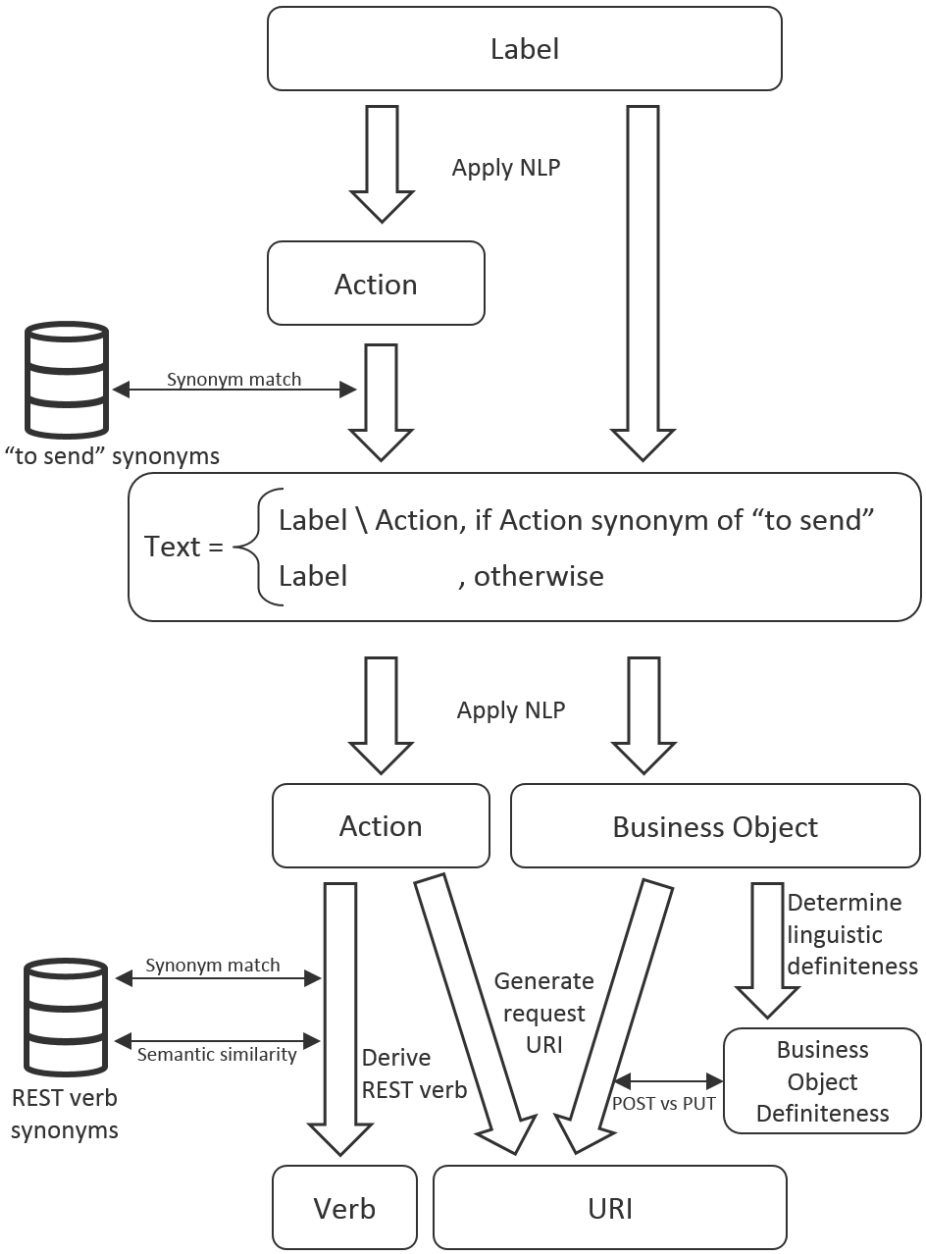

Figure 28: Overview of the semi-automatic derivation (advanced approach)

6.4 ADVANCED DERIVATION OF REST TASKS

In this section we introduce a modification of the core derivation approach to achieve better results with respect to the accuracy of the fully automatized derivation. The ultimate purpose is to have minimal intervention from the Web engineer when modeling a RESTful choreography. The advanced derivation approach is depicted in Figure 28.

Specifically we extend the core approach with two concepts. The first concept revolves about a certain set of choreography task labels that contain the action *to send* or a synonym thereof. As we will explain in

the first subsection these kind of actions may conceal the true meaningful action of the label. The second concept proposes a linguistic take on the difference between *POST* and *PUT*. The semantic feature *definiteness* of noun phrases [44] (commonly present in the family of Indo-European languages such as English) is used to decide between a *POST* or a *PUT* request. Namely, we show, in the second subsection, how the *definiteness* of the business object helps deciding between the two REST verbs.

6.4.1 *Choreography-specific labels*

With the introduction the label processing approach by Leopold et al. [39] in Section 6.2, we considered the send task as an instance of a activity task and applied the NLP approach to identify the action and the business object of the send task's label. However, we observed that in a choreography setting, where the choreography tasks are labeled based on their respective send tasks, the labeling style might conceal the intended action and business object.

Since choreography tasks represent a message which is sent across organizations, the labels tend to contain the action *to send* or a similar one, like *to enter*, *to submit*, *to mail*. All these action describe the passing of a message from one choreography participant to another. Putting these action asides, the labels are left with a single or multiple words, e.g., *order*, *invoice*, *request*, *application request*, *paper submission*, *short paper acceptance*. These label remainders sometimes hold their own action and business object, e.g., the remainder *paper submission* holds the action *to submit* and the business object *paper*, or the remainder *short paper acceptance* holds the action *to accept* and the business object *short paper*. When it comes to identifying the REST verb and resource these action-business object pairs are more insightful than the action *to send* or the like.

After all, all choreography task represent messages being sent from one participant to another. Even more, the action *to send* indicates no semantic preference towards any specific REST verbs since all REST requests are sent messages. For example, one can send a GET request for retrieving a resource representation or a DELETE request for deleting a resource. Therefore, we turn our focus on the remainder of the label containing the word *send* (or a similar one) to look for the additional information that can indicate an inclination of the label towards a specific REST verb.

Before checking whether the label contains the action *send* or the like we need to identify the verbs which can be used in a choreography to represent a message being sent. The set of synonym verbs of *send* is created and populated in a similar fashion as the REST synonym sets from Table 2. Hence, we have:

$$\mathrm{Syn}_{\mathrm{Send}} = \{\text{send, enter, submit, notify,} \underbrace{\text{mail, transmit, ...}}_{\text{derived}}\} \quad (4)$$

The first elements of the set are entered manually and the rest are automatically generated from the computational lexicons like WordNet.

If the label has an action that is element of $\mathrm{Syn}_{\mathrm{Send}}$, we analyze the remainder of the label as depicted in Figure 28. We re-apply the NLP approach on the remainder to identify a new action and business object. For example, the label *send review request*, *notify paper rejection*, *notify short paper acceptance* are reanalyzed as *review request*, *paper rejection* and *short paper acceptance*. Applying the same NLP approach yields the following action-business object pairs: (*request*,*review*); (*reject*,*paper*); (*accept*,*short paper*).

If, however, the label's remainder contains no new action and business object, we reset and simply apply the core derivation approach on the full label. For example in a label *send review*, we have α(send review) = *to send* and β(send review) = *review*. The remainder of the label is *review* after removing the action from the label. Applying the NLP approach on the remainder does not yield an action-business object pair. In this case, we consider *send* be the intended action and map it to the REST verb *PUT* (because *send* is part of the $\mathrm{Syn}_{\mathrm{PUT}}$). Hence the client sends a request to change the REST resource *review* to the state *sent*. The remainder does not need to be a single word to yield no action-business object pair. For example let us considering the label *send short paper*. Analyzing the remainder *short paper* would yield no new business object. In this case as well, we stick to the core derivation approach and map *send* to the REST verb *PUT* resulting in the REST request *PUT /short-paper/id/sent HTTP/1.1*.

6.4.2 *POST versus PUT*

We observed that the label's action is not the sole entity that can define the appropriate REST verb, especially when it comes to choosing between *POST* and *PUT*. Thus far we have considered *PUT* to be mainly used for modifying existing resources. *PUT* can, nevertheless, be used to create new resources as well, provided that the client "knows" the identifier of the resource. We take a moment here and discuss about what does it mean for the client to "know" about the identity of the resource at hand? Can we infer from the choreography label whether the client is familiar with the resource or not?

When we turn to linguistics, the answer to our questions can be found in the concept of *definiteness* [44], which is mainly associated to the semantic category of *identifiability* [6, 86]. Identifiability describes whether an entity is already introduced or identifiable in a discourse. In English, identifiability is usually (not always) expressed through determiners like the definite and indefinite articles, i.e., "the" and "a" or "an".

When we consider choreographies as a discourse/conversation between the participants we can infer from the task labels whether its

business object, which we map to the REST resource, is identified or familiar to the participants. We showed earlier in this chapter that the choreography labeling style is biased towards the initiator of the choreography task, i.e., the label expresses the action and the business object from the initiator's perspective. Therefore, we can conclude that the definiteness of the business object implies whether or not the initiator, or the client in REST terms, knows the REST resource identifier or not.

Due to the lack of a straightforward characterization of the definiteness [44], this thesis considers only the determiners. Although there are other type of determiners (like demonstratives, possessive determiners, quantifiers), we concentrate on the presence of the definite and indefinite articles that precede the label's business object. More precisely, we look at those cases where the label's action is mapped to *PUT* after applying our derivation approach. We perform a check for the presence of the English indefinite article "a" or "an". If indefinite article is present, we map the label's action to *POST* because it shows that the client is not aware of the resource identifier. For example, *submit a paper* task from Figure 26 is mapped to *POST /paper HTTP/1.1* since the client does not refer to a specific paper that familiar to both client and server at that point in the conversation.

In sum, our advanced derivation approach enhances the core derivation approach with an additional analysis of the choreography-specific labeling style and a linguistic approach to differentiate the REST verb *POST* from *PUT*. The former detects the presence of the action *to send* (and its synonyms) and looks for a more meaningful action in the remainder of the label in order to derive a more representative REST verb. The latter determines whether the client and server have identified already the REST resource by checking the definite or indefinite article of the data object from which the resource is generated. *POST* is used when the resource is not previously identified. Otherwise, *PUT* is chosen.

6.5 APPLICATION TO USE CASE

The final output of the derivation approach is a RESTful choreography. Figure 29 depicts the RESTful choreography diagram that is derived automatically by applying the advanced approach to *RMS* use case. The RESTful choreography of *RMS* describes the RESTful interactions between the REST API provider *RMS* and its clients: *Organizer*, *Author*, *Reviewer*. Therefore, all the choreography tasks whose recipient is *RMS* are translated into REST tasks. The remaining tasks are designated as email tasks. In our use case, out of 17 choreography tasks, 9 are REST tasks and 8 are email tasks. From the 9 choreography tasks, we encounter the following distribution of REST verbs: $2\times$*POST*; $5\times$*PUT*; $1\times$*GET*; and, $1\times$*DELETE*.

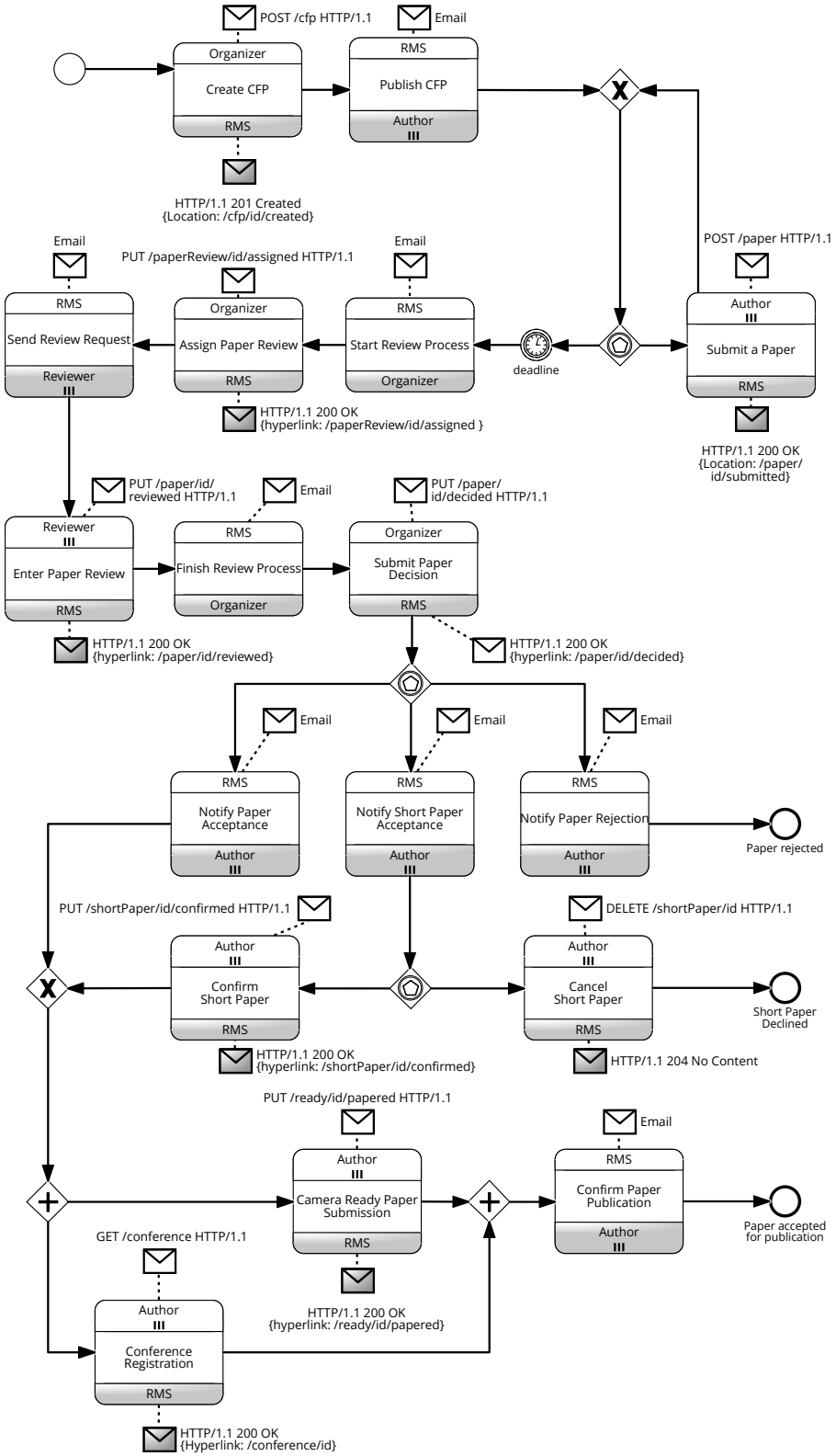

Figure 29: RESTful choreography for paper submission and review management

Out of 9 REST tasks, 5 are obtained by the core derivation approach and 4 by the advanced one. The labels *Enter Paper Review*, *Submit Paper Decision*, and *Camera ready paper Submission* are subject of the choreography-specific label analysis because their first acquired actions (i.e., *to enter,to submit,to submit* respectively) are elements of the $\mathrm{Syn}_{\mathrm{Send}}$. Their label's remainders (after removing the action) contain the following action-business object pair: (*review,paper*), (*decide, paper*), and (*paper, camera ready*). The *POST* versus *PUT* linguistic comparison effects only the derivation of the *Submit a Paper* label because of the presence of the indefinite article "a", which leads to the choice of using *POST* over *PUT*.

If we observe Figure 29 closely, not all the REST tasks are derived correctly. Two of the obtained REST verbs are the not desired output. They belong to the last two REST tasks—*Conference Registration* and *Camera Ready Paper Submission*. The derived REST request of the former is *GET /conference HTTP/1.1*. That is due to the action *to register* being an element of $\mathrm{Syn}_{\mathrm{GET}}$ because it is considered a synonym of *to read* from WordNet. The derived REST request from the latter is *PUT /cameraReady/id/papered HTTP/1.1*. Removing the first detected action *to submit*, which is element of $\mathrm{Syn}_{\mathrm{Send}}$, leaves us with the remainder *camera ready paper*. The application of the NLP approach on the remainder detects *to paper* as an action and *camera ready* as a business object. The desired action-business object pair would be (*submit, camera ready paper*) and the REST request would obviously be *PUT /cameraReadyPaper/id/submitted*.

Therefore, the intervention of the Web engineer is required to adjust or even re-specify the proper REST task. This is the reason the derivation approach presented in this chapter is considered a semi-automatic approach. Moreover, the Web engineer needs to enrich the email tasks with additional information, especially hyperlinks, to make the generated RESTful choreography complete with respect to REST specification (see Chapter 7).

6.6 SUMMARY AND DISCUSSION

The chapter defines a semi-automatic approach for deriving RESTful choreographies from BPMN choreography diagrams. The proposed approach is based on natural language analysis techniques to derive the most suitable REST verb for the interaction and to generate a REST request URI for the derived REST verb. Choreography-specific labeling style is taken into account. Our approach is evaluated by developing a REST Annotator tool and applying it to choreography diagrams from different domains. The details and results of the evaluation are presented in Chapter 10. The output of the tool was assessed by a REST expert. The verb identification is correct in 89.35% of cases, while the URI is correct in 93.65% of cases. This work contributes an additional

step towards the research gap between business process choreographies and their implementation.

7

RESTFUL CHOREOGRAPHY COMPLETENESS PROPERTIES

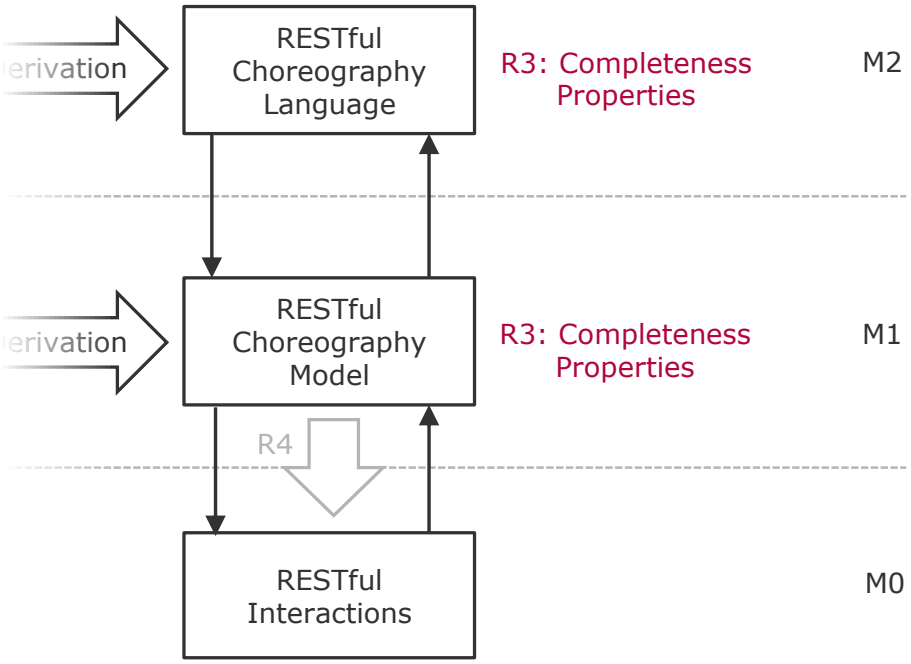

Figure 30: RESTful choreography completeness properties

In general, when models are refined towards the implementation level, preserving the consistency between the models (the high level model and the refined one) is mandatory for a successful mapping to an eventual implementation [21]. Whether a RESTful choreography is manually or semi-automatically derived from a process choreography, there is no guarantee, as it is, that the adding of REST information preserves the intended behavior of the choreography. More precisely a RESTful choreography model that is derived from a deadlock-free process choreography is not guaranteed to be deadlock-free. In this chapter, we show, with the aid of a use case, that a RESTful choreography can indeed reach a deadlock despite the process choreography it was derived from does not. This is due to the implicit behavioral aspects injected with the REST information. This chapter shows how we capture this REST-specific behavior explicitly in two formal properties that, if satisfied, ensure the absence of REST-induced deadlocks. These properties are hyperlink completeness and correct resource behavior.

In addition, the two proposed properties render a RESTful choreography complete with respect to REST constraints, up to the extent allowed by the the choreography's level of abstraction. Concretely, there are

REST constraints that are not fit for being considered in a choreography model, e.g., layered systems and cache (see Section 2.6). Hence, this chapter addresses requirement *R3: Completeness Properties*. The contribution presented here is partially based on the work by Nikaj and Weske [56].

7.1 MOTIVATION

To showcase the need for formal properties we introduce a RESTful choreography diagram in Figure 31 that models the organization of an online exam—from publishing to correction. The setting is that of a massive open online course (MOOC). The choreography is designed based on an interview with a lecture organizer in the openHPI[1] platform. We focus only on the online examination procedure taking place after all lectures are published.

The main participants in this choreography are the *teaching team*, the *MOOC platform*, and the *students*. The *teaching team* is responsible for publishing the exam and correcting the completed exams that are submitted by the *students*. The *MOOC platform* is a system that facilitates the interaction between the *teaching team* and the *students* by providing a Web interface and sending emails to coordinate the activities of the participants. Once the *teaching team* publishes the exam on the *MOOC platform*, the *students* are reminded by the latter via email. The *students*, then, may access the exam at any time before the deadline. In case the *students* access the exam, they have to submit it. The *teaching team* follows up with the exam correction and submits two possible outcomes into the *MOOC platform*. Either the exam is passed or not passed. If the exam is passed a *Record of Achievement* is created for the *students* to be accessed. Else, the *students* can retrieve a confirmation of participation, which can also be retrieved in the case that the *students* fail to access the exam before the deadline. As it can be observed from Figure 31, the exam choreography is composed of 7 REST tasks and 4 email tasks. The REST tasks represent requests sent by the *students* and the *teaching team* to the *MOOC platform*, and email tasks capture the notifications sent from *MOOC platform* to the *teaching team* and the *students*.

Judging from the control flow perspective, the presented RESTful choreography model does not contain any deadlock. However, when it comes to choreography models that is not sufficient for guaranteeing a successful execution. The choreography has to be enforceable [99]. A choreography cannot be enforced when the initiator of a choreography task is not aware of the direct preceding tasks. This means that the sequence flow between two tasks cannot be enacted because the initiator cannot know the right timing for executing the task. In the BPMN 2.0 standard this property is known as the *choreography activity sequencing*. This rule applies also to RESTful choreographies as an extension of

1 https://open.hpi.de/

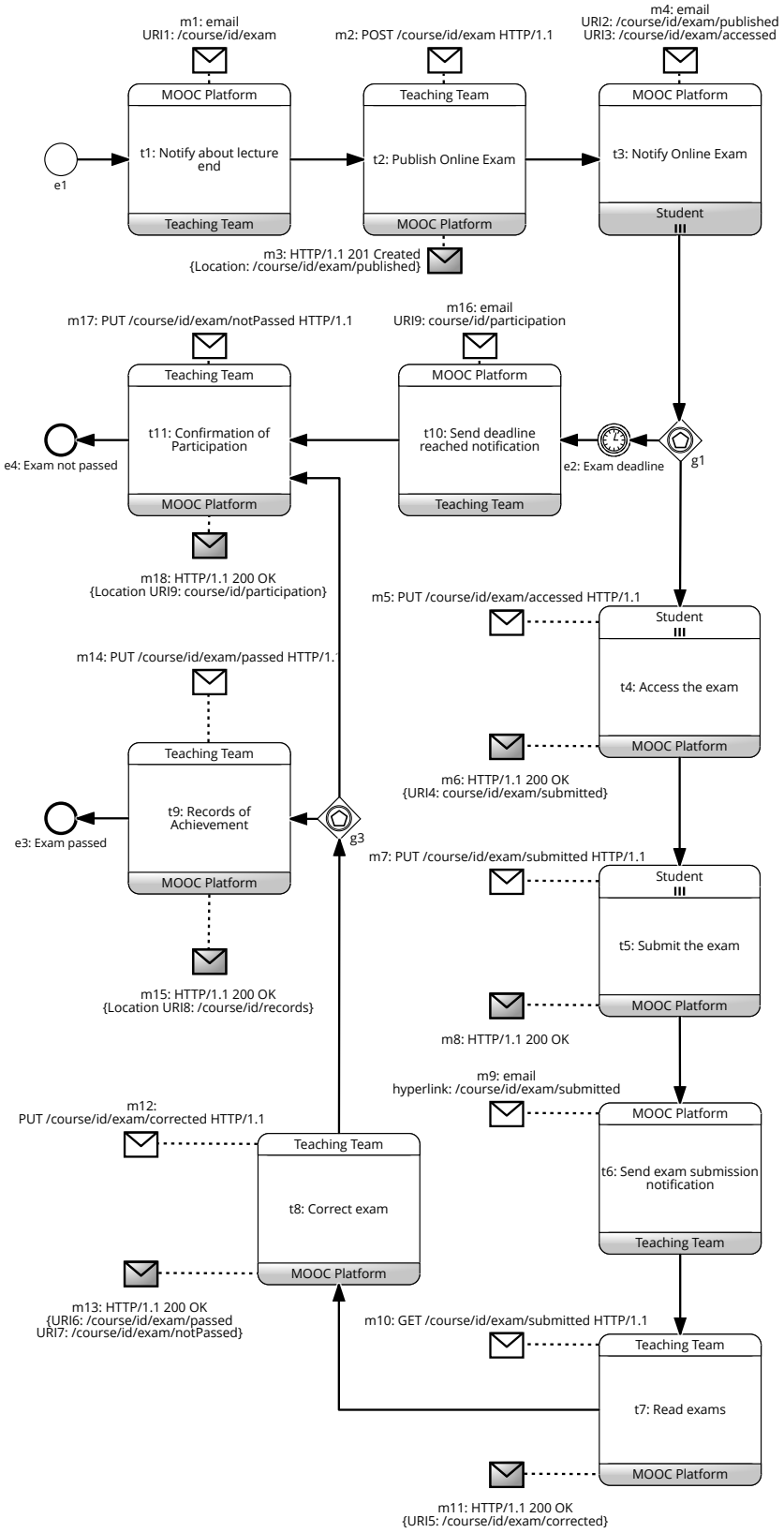

Figure 31: RESTful choreography of a MOOC exam

BPMN business process choreographies. Thus, it must hold true in order to enforce the RESTful interactions.

Since in this thesis we consider only the choreography task as an instance of a choreography activity[2], we formalize the RESTful choreography task sequencing property. First, we need to define a few helping notations. In the same way we defined $t\Box$ in subsection 5.2.2, we denote with $\Box t = \{t' \in T \mid ((t', t) \in S) \vee (\exists\, p_a = (t', n_1 ... n_k, t) \wedge \forall\, i = 1..k, n_i \in N \setminus T)\}$ the set of all choreography tasks that directly precede task t. Path p_a is defined in subsection 5.2.2. The gateways and events that precede the task t are ignored. Let us denote with $T^0 = \{t \in T \mid \Box t = \phi\}$ the set of all *starting choreography tasks*—they have no preceding tasks. Then, $T^* = T \setminus T^0$ is the set of all non-starting choreography tasks. Using these notations, the definition follows like this.

Definition 7.1 (RESTful choreography task sequencing)**.**
Given a RESTful choreography diagram $C_R = (N, S, P, M, U, V, etype, gtype, init, recip, initm, retm, mtype, verb, reqURI, server, hyperlink)$ and a participant $p \in P$, the basic rule of choreography task sequencing holds iff

$$\forall\, t \in T^*, p = init\,(t) \Rightarrow \forall\, t' \in \Box t, p = init(t') \vee p = recip(t')$$

◀

The exam choreography from Figure 31 is enforceable because the task sequencing holds true for every two consecutive tasks.

Nevertheless, enforceability does not suffice for a correct execution of the RESTful choreography. The REST-specific information injected into the choreography might lead to incorrect execution. For example, what would happen if the *students* are informed about the exam in task *t3* but no link is sent along for accessing it? From the process choreography perspective, this would not constitute a problem because neither the control flow changes nor the task sequencing rule is contested. However, the *students* cannot execute task *t4* because they miss the access hyperlink. Another aspect related to REST-specific problems is when a REST resource like *the exam* is addressed with the wrong verbs at the wrong order. For example, assume that in a RESTful choreography the exam is accessible via *GET* by the students after it can be removed from the server via *DELETE* by the *teaching team*. This unwanted behavior leads to a deadlock and goes against the REST principles defeating the purpose of RESTful choreographies in the first place.

In sum, two things can go wrong while executing RESTful choreographies: the hyperlinks needed for executing the REST requests are not provided upfront to the respective participants; and, the REST resources are addressed in an invalid manner or order. To avoid these particular problems, we introduce two formal properties that, if satisfied, ensure the absence of REST-induced deadlocks. The properties

2 A choreography activity is an abstract concept and its concrete instances are: choreography task, sub-choreography, and call choreography.

are hyperlink completeness and correct resource behavior. Hyperlink completeness asserts the HATEOAS principle of REST while correct resource behavior asserts the correct manipulation of resources, in that all REST methods/verbs are used accordingly.

When these two properties are satisfied, we consider the RESTful choreography to be complete with respect to REST constraints. We argue that other REST constraints, like cache and layered systems (see Section 2.6), are not appropriate for being represented at the level of abstraction and perspective that confine choreography models. The completeness property is defined as follows.

Definition 7.2 (RESTful Choreography Completeness)**.**
A RESTful choreography C_R is said to be complete, iff it is hyperlink complete and manifests a correct resource behavior. ◀

The next sections define in detail hyperlink completeness and correct resource behavior as well as methods for checking these properties.

7.2 HYPERLINK COMPLETENESS

In RESTful interactions, hyperlink is the client's main mean of navigation through communication with the server during the conversational flow. The only way to communicate with the server is by sending a request to a specific URI. As a response, the server provides the client with additional hyperlinks for the client to follow in future communications. Hence hyperlinks can be considered as a steering tool for guiding the RESTful interactions. For the server to client direction, an email communication is assumed. We do not explicitly take into consideration RESTful Push Interactions [77] because they are a special case of the normal RESTful interactions, i.e., to notify the client about new updates the role of server and clients are briefly exchanged.

A RESTful choreography is hyperlink complete if and only if all the URIs used in the REST requests are introduced previously to the clients in the form of hyperlinks. Naturally, the starting choreography tasks are excluded from this criteria because they have no preceding tasks. Hyperlink completeness also requires that all hyperlinks sent between participants are modelled in the RESTful choreography. We define hyperlink completeness as follows:

Definition 7.3 (Hyperlink completeness)**.**

A RESTful choreography is hyperlink complete iff $\forall\, t \in T^*$ the initiating participant $p \in P$ sends a REST request via URI $u \in U$ in task t, then for all execution paths leading to task t the request URI u is passed to participant p embedded in a response or email message.

◀

7.2.1 *Structural hyperlink completeness*

Tracking the execution paths in a choreography model becomes cumbersome in the presence of concurrent execution. For convenience, we propose additionally a simpler version of the hyperlink completeness property that is defined on the model structure and is only applicable on models that contain no parallel tasks (or when the parallel tasks are replaced with with a tree of exclusive paths that capture the same exact behavior). In this case, we do not need the execution paths but just the model paths that start at the beginning of the choreography. We define this property formally as follows:

Definition 7.4 (Structural hyperlink completeness)**.**

A RESTful choreography is structurally hyperlink complete iff
$\forall t \in T^*, mtype(initm(t)) = req \Rightarrow \forall t^0 \in T^0, \forall p_a = (t_0, .., t), \exists t' \in p_a(t^0, .., t) \mid (init(t) = init(t') \wedge reqURI(req(t)) \in hyperlink(retm(t')))$
$\vee (init(t) = recip(t') \wedge reqURI(req(t)) \in hyperlink(initm(t')))$

◀

This property should not be used for RESTful choreographies that manifest concurrent execution because it is too strong, i.e., the hyperlinks are required to be passed in all parallel paths while, in reality, in only one path is sufficient. The structural hyperlink completeness is stronger than the hyperlink completeness because the latter implies the former but not vice-versa.

7.2.2 *Checking hyperlink completeness*

Nevertheless, we propose a method for checking the hyperlink completeness of any RESTful choreography, including those who manifest concurrency. The method is based on translating the RESTful choreography into a Petri net model [80], which, due to its formal semantics, is subject to a plethora of business process analysis [43]. The transformation to Petri nets follows a two step approach.

1. *Generate the control flow.* In this step we employ the derivation rules from Dijkman et al. [45] to capture the control flow of the choreography in a Petri net model. Although the derivation rules are originally proposed for deriving Petri nets from business process models, choreography diagrams use a subset of the process diagram elements where process activities are replaced with choreography activities. The output Petri net $PN = (P_n, T_n, F_n)$[3] is composed of: a set of control flow places $P_{cf} \subset P_n$; transitions that represent choreography tasks $T_t \subset T_n$; and, silent transitions that are used for implementing the behavior of certain choreography gateways.

3 the subscript "$_n$" is used to separate the Petri nets places from the participants

2. *Introduce information places.* The information place stands for the participant's awareness of a certain URI. Given a Petri net PN, an information place $p_i \in P_n, p_i = (p, u)$ represents the pair of a participant $p \in P$ and a URI $u \in U$. If at least a single token is present in the information place $p_i = (p, u)$, then the participant p is aware of the URI's u existence. The information places are added to the Petri net P_n according to the following transformations:

   - *REST request message.* For every REST request in the RESTful choreography model, add the information place $p_i = (p, u)$, as an input and output place of the respective transition $t_n \in T_n$, where p is the requesting participant and u is the request URI like shown in Figure 32. The token in this information place is required for the transition to fire but it is not ultimately consumed because the requesting participant does not loose awareness of the request URI after the task t is executed. When the request has m hyperlinks embedded in its body (usually via a *PUT* or *POST* request), add $m \in \mathbb{N}$ number of information places $p_i = (p', u_k), k = 1..m$, as output places of the respective transition $t_n \in T_n$, where p' is the recipient of the REST request. When task t is executed, the participant p' is made aware of all hyperlinks embedded in the rest request message. This case is not depicted in Figure 32 to avoid clutter. However, it is very similar to the email message explained above and depicted in Figure 33.
   - *REST response message.* For every REST response in the RESTful choreography model, add $n \in \mathbb{N}$ number of information places $p_i = (p, u_j), j = 1..n$, as output places of the respective transition $t_n \in T_n$, where n is the number of distinct hyperlinks embedded in the response message and p is the recipient of the response message (see Figure 32). When task t is executed, the participant p is made aware of all hyperlinks embedded in the response message.
   - *Email message.* For every email message in the RESTful choreography model, add $m \in \mathbb{N}$ number of information places $p_i = (p, u_k), k = 1..m$, as output places of the respective transition $t_n \in T_n$, where m is the number of distinct hyperlinks embedded in the email message and p is the recipient of the email (see Figure 33). When task t is executed, the participant p is made aware of all hyperlinks embedded in the email message.

The resulting Petri net captures the choreography's control flow and hyperlink behavior. Therefore, assuming that the source RESTful choreography is enforceable and there is no control-flow deadlock, we have: The RESTful choreography is hyperlink complete iff the resulting Petri

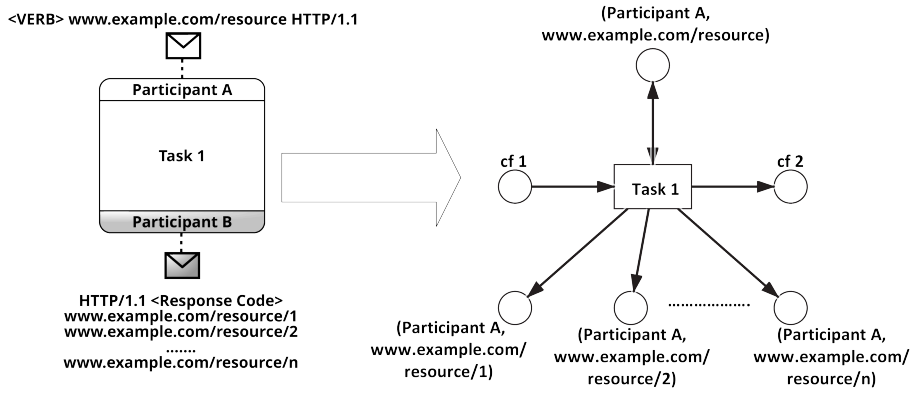

Figure 32: Mapping of a REST task (REST request plus REST response) to Petri net

net is deadlock free. That is due to deadlocks being introduced by either the control flow or the missing hyperlinks that block the choreography tasks from being executed.

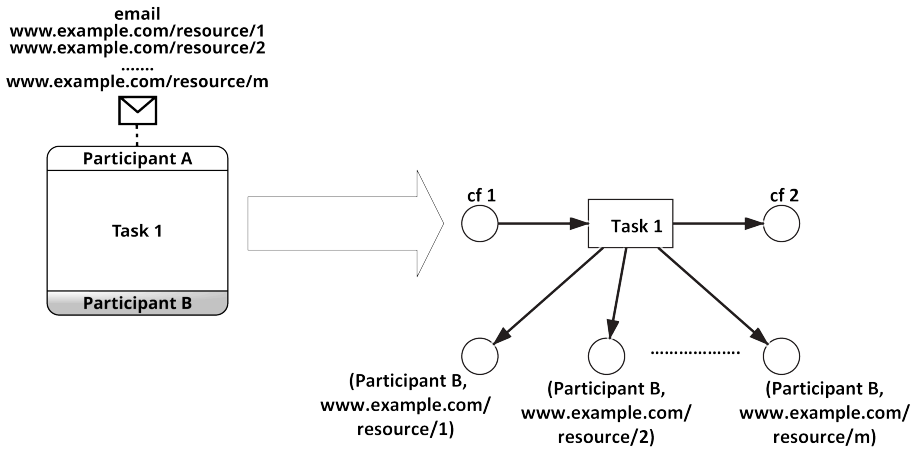

Figure 33: Mapping of a email task (email request) to Petri net

LOOPS The resulting Petri net may be unbounded in the presence of loops. When an URI is passed to a participant a token is added in the respective information place. If the choreography task is part of a loop than infinitely many tokens may be added to the information place. The number of tokens will never be reduced because any time the participants uses the token it writes it back (see Figure 32). This does not constitute a problem for our approach but it makes the check of deadlocks not straight forward, e.g., checking for soundness [91] via recheability graphs on the resulting unbounded Petri net is not possible. However, the unboundedness problem can be solved by applying the minimal coverability graph approach by Finkel [24] to the resulting Petri net. The transformation to Petri net is applied to our online exam use case below Section 7.4.

7.3 CORRECT RESOURCE BEHAVIOR

The second property is about checking the behavior of REST resources, i.e., checking whether or not the resources involved in the choreography behave as expected. Defining this property assures the users of the RESTful choreography that each REST resource does not undergo undesired behavior. This is particularly useful in the case of RESTful choreographies due to many participants accessing common resources, e.g., the resource *exam* is accessed by the *teaching team* and the *students* multiple times in the choreography.

Definition 7.5 (Behavioral Correctness)**.**
A RESTful choreography manifests a correct resource behavior if all involved REST resources behave correctly. A resource is said to behave correctly if it:

- is created with a POST /resources or PUT /resources/id
- changes its state with a PUT /resources/id/newState
- is accessed with a GET /resources/id/State yielding no state change
- is deleted with a DELETE /resource/id
- can only be accessed or modified after it is created and before it is deleted.

◀

Notice that these conditions apply only when the change of the resource state is triggered by a REST request. The resource state can also change internally by the server. In this case, we cannot enforce rules as it is out of the interaction scope and it depends on the server's application logic.

To check the behavior of the resource, the resource states are derived in the form of a UML state machine [63] from the RESTful choreography. The derivation is performed for every REST resource found in the choreography. The correctness of the resource behavior can than be easily checked on the state transition model. If all transitions of the derived state machine comply with the transitions from Figure 34 then the resource behaves correctly. Figure 34 is the graphical representation of resource correctness rules presented above.

The derivation procedure of the resource behavior starts with isolating a resource in the choreography diagram, e.g., the resource *exam* in our running example. Then, the REST tasks that are irrelevant to the chosen resource as well as email tasks are replaced with a sequence flow. Same is done with all the intermediate events. The gateways are kept untouched because they are needed to determine alternative paths during state transitions of the resource. At last, we have a RESTful choreography which contains only REST tasks addressing only a single resource. Subsequently, we transform the RESTful choreography into

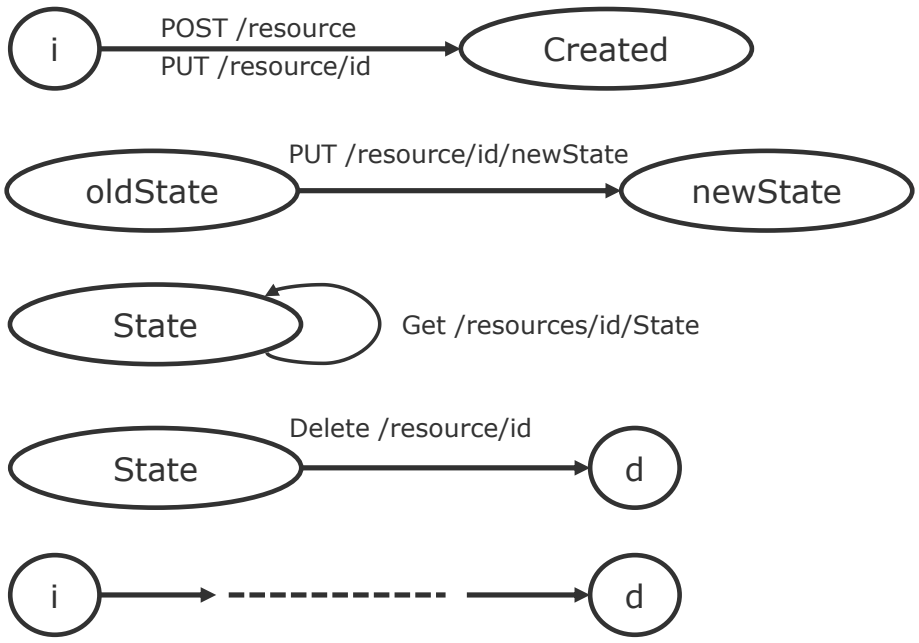

Figure 34: Correct resource behavior

a state transition diagram. The state labeling is based on the state of the REST resource, e.g., the exam is in the state published, accessed or submitted. State changes inducted by the REST request are labeled in the state transition diagram with the corresponding REST request message like shown in Figure 34.

7.4 APPLICATION TO USE CASE

In this section we check the defined formal properties on the RESTful choreography depicted in Figure 31 and argue about its RESTful completeness.

CHECKING HYPERLINK COMPLETENESS. The hyperlink completeness property of this choreography is checked by generating the respective Petri net following the derivation rules introduced above. The resulting Petri net is depicted in Figure 35. The participant names are abbreviated to *tt* and *s* for the *teaching team* and *students* respectively. Following the execution path where the students participate in the exam we can observe that there are no deadlocks—neither concerning the control flow, nor the passing of the hyperlinks. However, when we observe the alternative execution path that models the case where a student does not participate in the exam, we observe a deadlock in task *t11*. The deadlock is not caused by the control flow because the control flow place *cf9* is reachable through any execution path starting from place *cf1*. The deadlock is caused by the lack of token in place *tt:/course/id/exam/notPassed*. This means that the *teaching team* is not

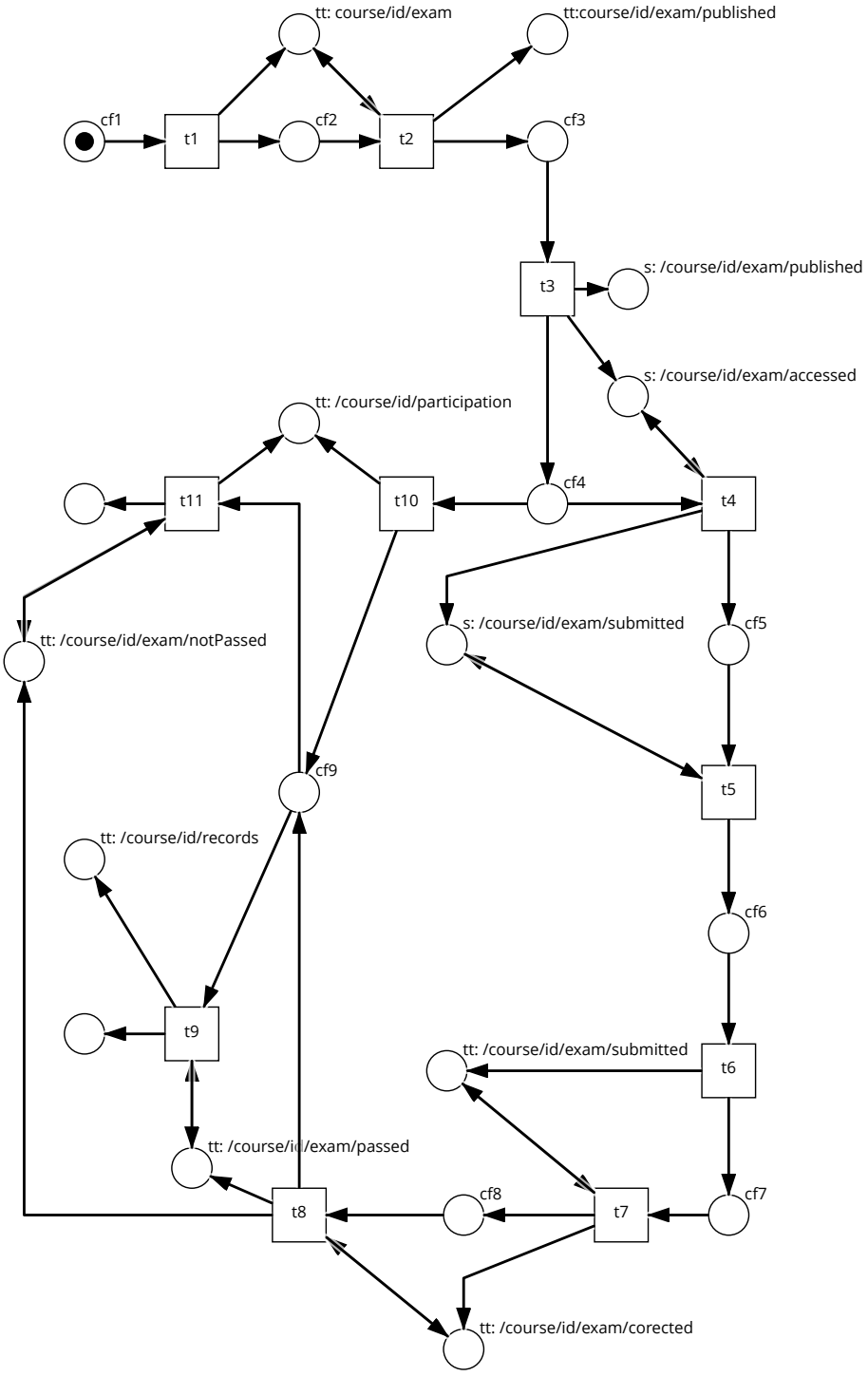

Figure 35: The generated Petri net from the RESTful choreography of online exam

aware of the hyperlink */course/id/exam/notPassed* when the left branch of the choreography is taken. Therefore, the RESTful choreography of the online exam is not hyperlink complete. In order to fix the problem, the hyperlink responsible for changing the *exam* state to *not passed* has to be provided to the *teaching team* also in the case the students do not participate in the exam.

CHECKING RESOURCE BEHAVIOR CORRECTNESS. For checking the resource behavioral correctness of the use case, we identify one REST resource, i.e., the *exam*. Figure 36 depicts the lifecycle of the *exam* resource derived in the manner described in the previous section. Following the definition of resource behavioral correctness we can conclude from Figure 36 that the *exam* resource has a correct behavior because it is: created via a POST; accessed via GET and no new states are introduced; edited via PUT leading to a new states; and finally, all requests are performed after the resource is created and before it is deleted (in this example there is no DELETE request). Since the MOOC choreography has only one main REST resource that has a correct behavior, we conclude that the choreography manifests a correct resource behavior. Additionally, deriving the state transition of the resources helps the developers of RESTful APIs to understand the allowed interactions, e.g., it is not allowed to have a *PUT /course/id/exam/accessed* after *PUT /course/id/exam/notPassed*. *GET* requests can be easily checked if they are safe (i.e., do not introduce side effects) by making sure that every *GET* state transition is looped around a single state.

As conclusion, the RESTful choreography of the online exam manifests a correct resource behavior but it is not hyperlink complete, thus, making it not RESTful complete.

7.5 SUMMARY

In this chapter we propose formal properties of RESTful choreographies, namely: hyperlink completeness and behavioral correctness. The former assures that all the hyperlinks used by the clients for sending REST requests are provided to them prior to the request occurrences. The latter makes sure that all REST resources behave as expected and according to REST principles during their lifecycle. When both these properties are satisfied we consider the RESTful choreography to be complete with respect to REST constraints.

For checking hyperlink completeness, we translate RESTful choreography models into hyperlink-aware Petri nets where we perform behavioral analysis for detecting deadlocks. While the resource behavioral correctness is checked by making use of a different view—a state transition diagram that is derived from the RESTful choreography for each REST resource involved in the interaction. This view provides to Web engineers a clearer perspective on each resource behavior by

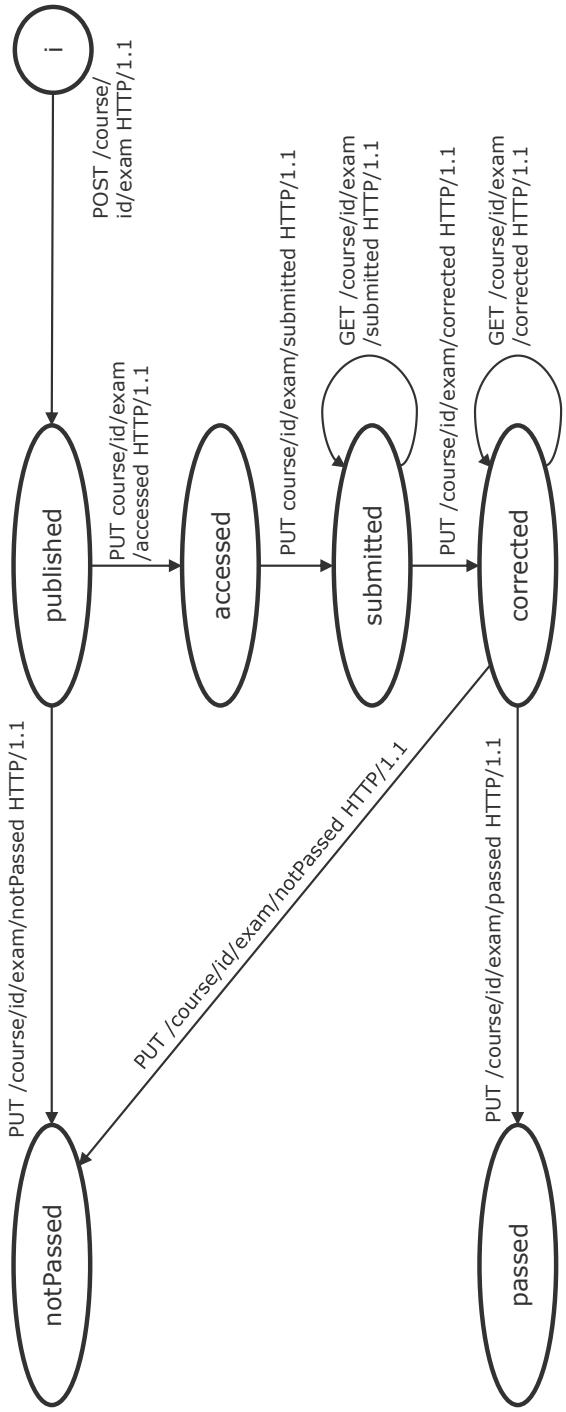

Figure 36: Exam lifecycle derived from the RESTful choreography in Figure 31

emphasizing the state transitions induced by REST requests. This is particularly useful because it provides an overview over the allowed resource changes at any point during the RESTful interactions.

Part III

FROM RESTFUL CHOREOGRAPHIES TOWARDS RESTFUL INTERACTIONS

8

REST-ENABLED DECISION MAKING IN BUSINESS PROCESS CHOREOGRAPHIES

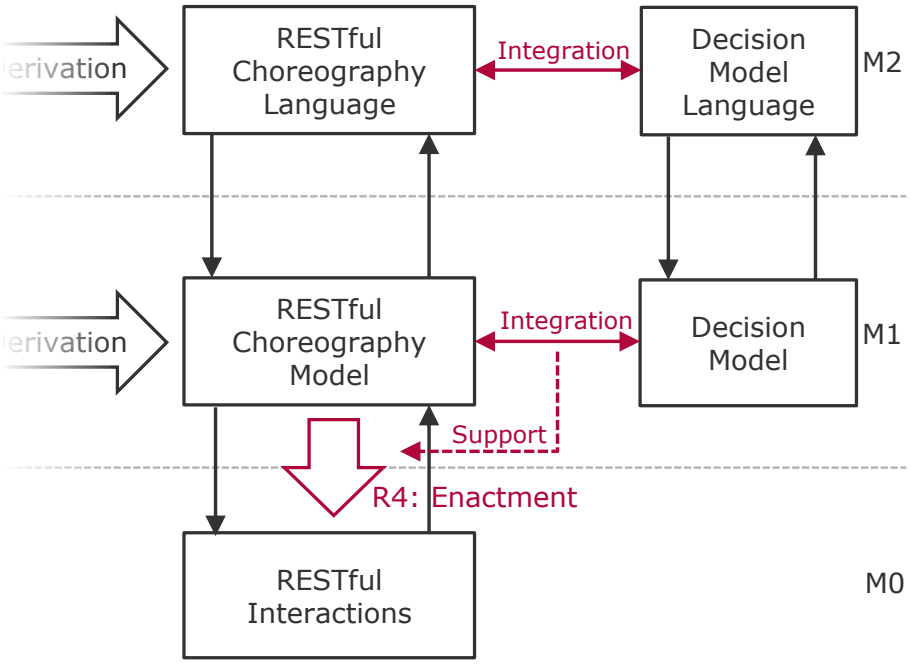

Figure 37: REST-enabled decisions in business process choreographies

Implementing choreographies in general, and RESTful choreographies in particular, remains a challenge due to the lack of a central locus of control. The successful implementation of the choreographies, like with any type of contract, depends solely on the fact that the involved participants act as prescribed in the choreography (contract). In other words, each participant has to autonomously execute their business processes in a way that they jointly realize the choreography. Not only do the participants have to execute their processes properly, but they also have to have a similar understanding of the choreography in general and the data that is being exchanged in particular—especially the data that affect the choreography's control flow. The control flow branching in choreographies comes as a result of some decision making, which is reflected in the choreographies' exclusive gateways. The peculiar implementation of such gateways is the main subject of investigation in this chapter, the contribution of which is considered as a partial fulfillment of requirement *R4: Facilitating the enactment of RESTful choreographies*.

Since there is no central mechanism of control, participants may interpret the data relevant for decision making and the decision making itself differently. To tackle this problem, this chapter offers a solution

that delegates the decision making to a decision service. The service is provided to the choreography participants via a REST interface and it is based on the recently published Decision Model and Notation (DMN) standard [67], hence, making the decision unambiguous. We introduce the concept of RESTful decision service and how this service is used in conjunction with RESTful choreographies. The RESTful decision service assures a correct implementation of choreographies' exclusive gateways and provides a blueprint for RESTful services that offer decision-making solutions based on the DMN standard. This chapter's contribution is, for the most part, based on our work in [56].

8.1 PROBLEM STATEMENT

BPMN 2.0 business process choreography models borrow the gateway constructs from the process models. Gateways are used for modeling alternative or parallel paths in the control flow. In process models, the central orchestrator (e.g., a business process engine) is responsible for evaluating the gateways and determining the execution path accordingly. However, choreographies lack a central mechanism of control, thus, allowing the use of gateways only under a set of constraints. Below, we focus on the particular problem of implementing choreographies' exclusive gateways.

8.1.1 *Choreographies' exclusive gateway constraints*

Figure 38 depicts an example of a choreography diagram that describes the interaction of a manufacturer with its customer and suppliers. A customer sends an order request to the manufacturer. If the manufacturer has the necessary parts for manufacturing the product, it delivers the product to the customer. Otherwise, the manufacturer makes a request for tender to different suppliers for the product part it needs. The suppliers follow up by sending their offers. After receiving all offers, the manufacturer announces the score which represents the level of satisfaction for each supplier's offer. If there is at least one single score which passes a threshold then the supplier with the best score receives the payment from the manufacturer and sends the product part. Otherwise, the tender is not successful and the suppliers are asked to send again their offers. The tender can be closed at anytime during this loop but it is not shown explicitly in the choreography model for simplification purposes. If there is a winning offer, the manufacturer receives the product part that are necessary to manufacture the final product. The final product is, then, delivered to the customer, who in turn sends the payment. In this scenario, our point of interest is the exclusive gateway that decides the winner of the tender or the lack thereof.

In choreography diagrams, exclusive split gateways model alternative paths. Exclusive join gateways are simply used to join the control flow without any synchronization requirements. Therefore, they are

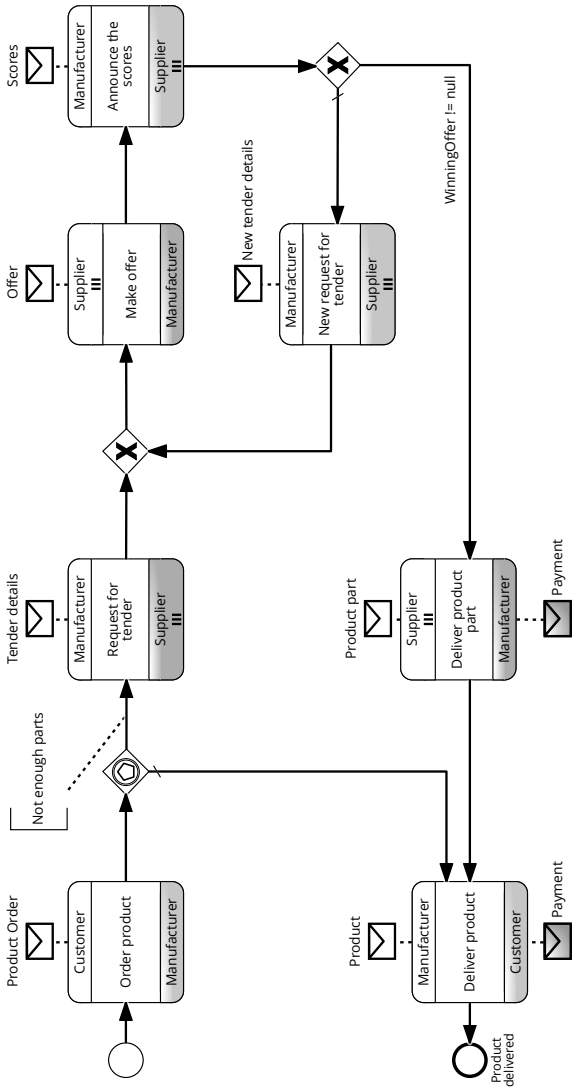

Figure 38: A business process choreography model for organizing a tender

not the focus of our chapter. Here, by the term *exclusive gateway* we refer to only the exclusive split gateway. Choreographies' exclusive gateways are constrained in their usage compared to their respective counterparts in business processes. In order to implement a choreography the following three constraints should hold [65].

- The data used as basis for the gateway conditions must have been sent via a message at some point in the choreography before the gateway. The message(s) containing the data is sent or received by all participants that are affected by the gateway.
- Any change of the data must be visible to all the involved participants.
- Every participant must interpret the data in the same way.

However, there is a problem in implementing such constraints when it comes to the interaction of participants whose role can be filled by many possible business actors. The problem consists in that diverse business actors can have different understandings of the data used for the decision making, leading the choreography to be out of sync. This is due to the fact that the business actors might be very diverse in terms of, e.g., domain, culture, and country. In the part procurement scenario from Figure 38, let us consider the constraints related to the exclusive gateway that leads to either repeating the tender or declaring the winner. Two out of three constraints are considered to be satisfied. The data used for the gateway conditions is sent from manufacturer to the supplier (the participants that are affected by the gateway) via the *Score* message. Moreover, the decision is executed immediately after the data is shared. Therefore, it can be assumed that the data has not changed between the moment it is shared and the moment the decision is made. However, since the role of the supplier can be taken by a diverse number of companies from different parts of the world we cannot ensure that all the participants have the same understanding of the shared data and the way the decision is made. Therefore, we need a precise way for conveying the decision.

8.1.2 *Decision Model and Notation*

In order to make the decision less ambiguous for all participants we use the DMN standard [67]. DMN provides two levels for capturing the decision into models: the decision requirement level and the decision logic level. The former provides an overview over the decisions' relation to their inputs and each other. The latter describes the exact decision logic for each decision. For expressing the decision logic in a formal way, DMN provides, among other ways, the Friendly Enough Expression Language (FEEL). Figure 39 shows an example decision model: *decisions* are rectangles; *input data* are ellipsis; *information requirement edges* are solid; and *knowledge sources* are rectangles with a wavy bottom. The decision element is associated with a FEEL expression displayed as an annotation next to it.

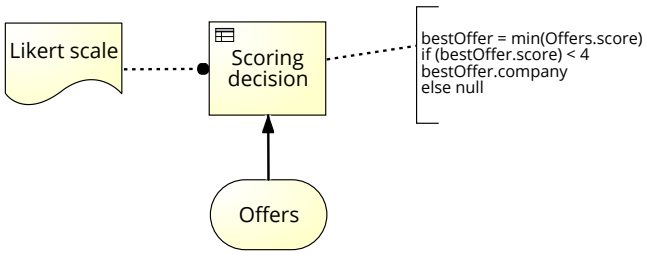

Figure 39: Decision model used by the manufacturer to decide on a supplier (the right-most gateway in Figure 38)

The example from Figure 38 is based on the tender procedure choreography. More particularly, this is the decision model employed by the manufacturer to decide which supplier to choose from all the suppliers that made an offer. The decision element is labeled *Scoring decision* and takes as input data a list of *Offers*. The *Offers* have two attributes: *company* and *score*, where *company* represents the name of the company that made the offer and *score* represents the assigned score by manufacturer to the offer at hand. There are two possible outcomes: either there is no winning offer such that a new tender is requested, or one of the offers was chosen and the product parts are purchased from the respective supplier. The fact whether or not there is a winning offer depends on the scores, which can range from 1 to 5, according to the well known Likert scale. This scale says that the lower the score, the better. Therefore, the FEEL expression associated with the decision element first determines the offer with the minimum score. Then it checks if the score is less than four. If that is the case, the respective company's name is returned. Otherwise, the decision yields a *null* value.

As we can infer from above, the standard decision model leaves no room for misinterpretation. Nevertheless, an additional problem arises: the DMN model needs to be fully integrated into the choreography model without introducing undesired side effects like, for example, rendering the choreography unenforceable.

8.2 RESTFUL DECISION SERVICE FOR CHOREOGRAPHIES

At this point, where we showed that the choreography's decision logic can be captured to a DMN decision model, we need to integrate it with the choreography diagram. We propose a solution for encapsulating the decision model into a decision service that can be called by the involved participants of the choreography diagram to aid the implementation of the exclusive gateway. We choose the REST architectural style [23] due to its prominent presence when it comes to the design of Web application interfaces. Our proposed solution is the design of a RESTful decision services that is used in tandem with a choreography model. This is where the RESTful choreographies come into play. The RESTful service is integrated into a partial RESTful choreography model to achieve a correct implementation of the choreography's decisions (exclusive gateway). It is called partial RESTful choreography because we introduce REST elements only for the choreography part that is concerned with the RESTful communication of the participants with the decision service.

In the next subsection the interface of a DMN-based RESTful service is specified. The integration of the service into a RESTful choreography is described in the subsequent subsection.

8.2.1 *REST interface of decision services*

To solve the problem of the data misinterpretation we propose a RESTful decision service that is built on the DMN standard. The RESTful decision service provides a REST interface for creating, managing, and executing shared decisions. The decision owner is responsible for creating the decision logic and providing it to the decision service. Thus, we have a single decision for all the participants to execute. The decision service is responsible for computing the decision each time it is called from the participants who are affected by the exclusive gateway. The output of the decision should comply to the conditions of the sequence flows originating from the exclusive gateway, hence, making it trivial for the other participants to relate the decision output with the correct path in the choreography.

For creating the RESTful API of the decision service, we map a REST interface to the main concepts of DMN. As explained in the previous section, these main concepts are the decision logic, inputs, outputs and execution. The REST interface is designed according to RESTful design rules [46] and design patterns [73]. A RESTful decision service provides the REST interface described in Table 5.

The creation of a new decision logic is the only action that is not included in the choreography diagram because it is about deploying the decision model on the server. The decision owner is the participant of the choreography that is responsible for creating the decision logic like the manufacturer in Figure 38. Nonetheless, the decision model, similarly to the choreography model, can be designed in collaboration with other participants. The RESTful choreography diagram models the interaction that the participants have with a particular decision logic, e.g., *Scoring decision* from Figure 39. Hence, deploying the decision logic model via *PUT /decisionName* on the RESTful decision service is a prerequisite for executing the choreography.

Anytime a new instance of the choreography gateway has to be executed, a new decision instance that corresponds to that gateway instance should be created by the decision owner, e.g., a new scoring decision id should be created for any tender that is requested and eventually decided at the gateway. It is important to distinguish between different decision instances because the participants should have access only to the decision instance that affects their behavior. The mapping is not always one gateway instance to one decision model instance. In case of a loop a new decision can be instantiated many times. For example, when *winningOffer* = *null* a new tender is created and, therefore, a new decision instance has to be created. In this case the participants should distinguish the new tender from the old one from the change in the decision URI.

The decision owner inserts the decision inputs after the decision instance is created. Providing the decision inputs completes all the requirements for the execution of the decision. This makes the decision

Table 5: The interface of a RESTful decision service

Decision Action	**REST Request and Response**
Create a new decision (logic)	⇒ PUT /decisionName
	⇐ HTTP/1.1 201 Created {Location: /decisionName}
Create a decision instance	⇒ POST /decisionName/
	⇐ HTTP/1.1 201 Created {Location: /decisionName/id link: /decisionName/id/inputs link: /decisionName/id/execute}
Insert the decision inputs	⇒ PUT /decisionName/id/ inputs/inputName
	⇐ HTTP/1.1 200 OK {link: /decisionName/id/ inputs/inputName}
Read the decision inputs	⇒ GET /decisionName/id/ inputs/inputName
	⇐ HTTP/1.1 200 OK {link: /decisionName/id/ inputs/inputName}
Execute the decision	⇒ PUT /decisionName/id/execute
	⇐ HTTP/1.1 200 OK {link: /decisionName/id/output}

service available for execution by any participant that knows the decision URI. The participants can optionally read the inputs before executing the decision. However not every participant is allowed to change the input. This would lead to unintended paths in the choreography. The participants responsible for editing the input can be identified in the choreography diagram because the input data originates from them. For example, the choreography diagram in Figure 38 shows that the manufacturer announces the scores and, hence, is responsible for creating or editing of the input data used in the decision. When a request is sent for the execution of the decision, the response provides the output

of the decision. The output of the decision should be consistent with the conditional sequence flows that follow the exclusive gateway.

8.2.2 *Integrating RESTful decision services into choreographies*

Having a REST interface for the decision service is not sufficient. The objective is to go from a business process choreography to a partial RESTful choreography that solves the exclusive gateway problem by incorporating the RESTful decision service. A RESTful choreography fragment that secures the communication with the RESTful decision service is inserted into the existing choreography. In this section, we provide a stepwise method for embedding decision services into choreographies.

Since a part of the choreography is transformed into a RESTful choreography fragment, it must be guaranteed that the introduction of the new elements does not impact the correct execution of the choreography, assuming already that the initial business process choreography is enforceable. To this end, we consider two main properties: *the task sequencing* property stating that the initiator of each choreography task is either initiator or recipient in the direct preceding tasks. The very first choreography task is an exception since it does not have any preceding task (see Definition 7.1); *Hyperlink completeness* stating that all REST request URIs used in the RESTful choreography are provided to the initiator at some point upstream in the choreography. An exception is made for the first occurring REST request (see Definition 7.3).

The decision service can be an external participant, i.e., an additional business actor who provides decision services or hosted by one of the participants. In terms of the REST interface, that is irrelevant because only the Web domain would be different and the rest of the URI would not change. For the remainder of this paper, we assume the more complex and interesting case where the decision server is an external participant in the RESTful choreography.

The stepwise method for embedding the RESTful decision service in business process choreography is given below. As mentioned above, the only requirement to start this method is the creation of the decision logic by the decision owner. The method consists of the following 5 steps:

1. Locate the choreography task where the decision-relevant data is passed for the first time. Add before the located task a new RESTful task that creates a new decision instance. The response should contain the location of the instance, the URI of the decision input, and the URI of the decision execution. The initiator should be the decision owner.

2. Next to the newly added task, add a new RESTful task that inserts the decision inputs using the link passed from the previous task. Again, the initiator should be the decision owner.

3. Change to email task every task that models the passing of the decision-relevant data. Replace the content of the email message with information that describes the location of the decision together with the inputs and execution links.

4. Locate the exclusive gateway. Add before the gateway as many RESTful tasks as participants (affected by the gateway) in parallel. Each participant individually sends a request for executing the decision to the decision service and receives the output of the decision service as a response.

5. Between the exclusive gateway and each immediate following task insert an email task, where the initiator is the decision service and the recipient is the same as the initiator of the direct following task. The role of this task is to inform each following participant (the ones that directly follow the gateway) about the fact that all the other involved participants have executed the decision and that she or he should proceed accordingly.

Let us analyze whether the task sequencing properties is preserved with every step taken:

- *Step 1* The insertion of the new RESTful task does not break the property because the initiator remains the same.

- *Step 2* The subsequent RESTful task added has again the same initiator. This means that the property is preserved because the initiator is present in the direct preceding task.

- *Step 3* In this step, there is no change in the participants of each choreography task. Only the message content has been changed.

- *Steps 4 and 5* The BPMN specification requires that the initiators of the choreography task following the gateway have to part of the choreography task that directly precedes the gateway. Adding parallel REST request that execute the decision service in step 4 does not introduce any breach of the property because the initiators are those affected by the gateway. The recipient on each REST task is the decision service. The decision service is also the initiator of the email task added directly after the gateway in step 5. Hence, the requirement of the choreography exclusive gateway is satisfied.

As a conclusion, the transition from a choreography diagram to a partial RESTful choreography diagram (hosting a decision service) does not break the task sequencing property of the diagram.

Regarding the hyperlink completeness property, we show that it is preserved. The link used for the first time in step 1 is the first occurring link. The concrete decision instance link, inputs link and execution link

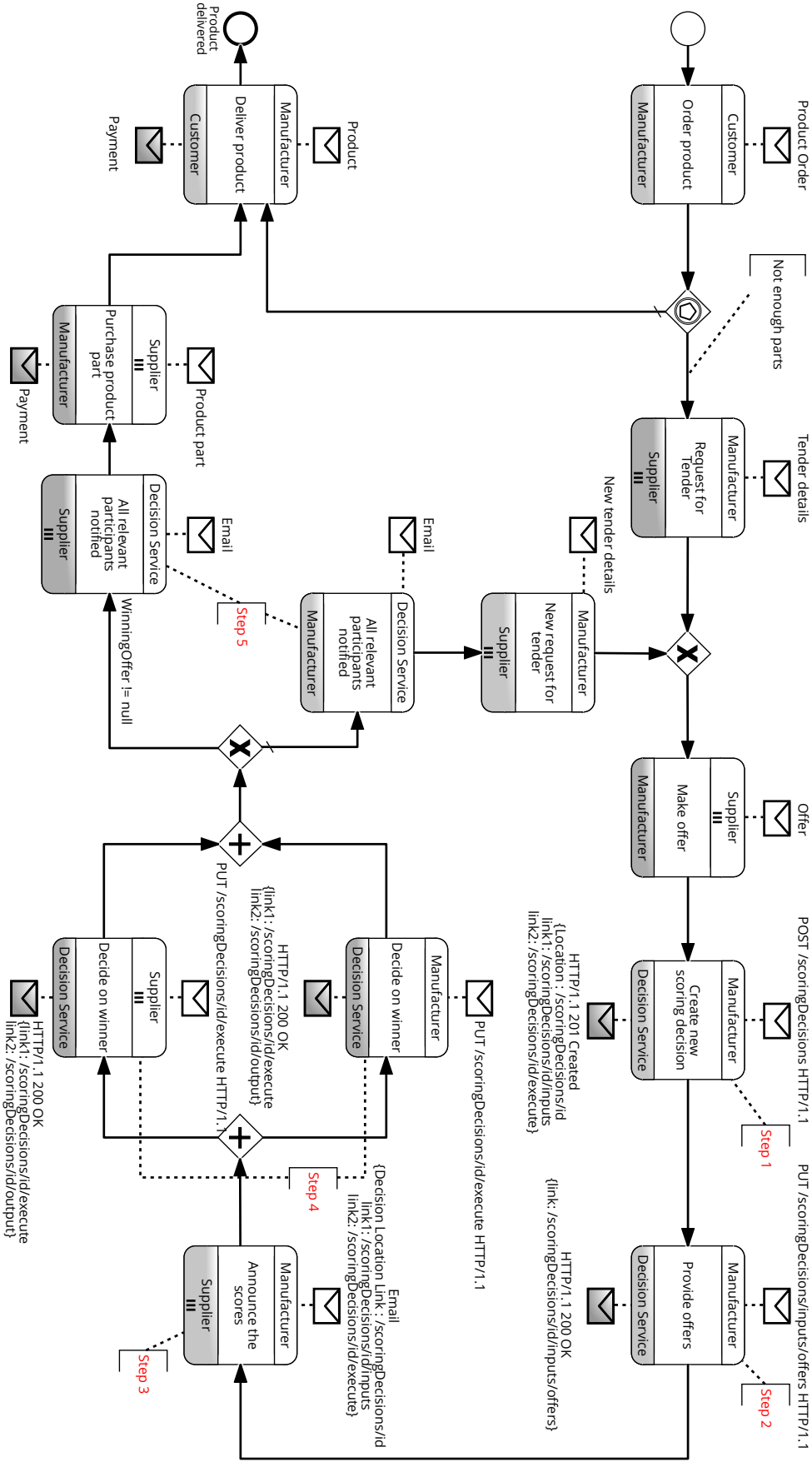

Figure 40: A partial RESTful choreography model for organizing a tender with the assist of a RESTful decision service

are provided to the decision owner via the response of the REST call from step 1. Then, the decision owner enters a new input using the inputs link in step 2. In step 3, all links introduced before are provided to the participants. Eventually, the participants use these links to execute the decision in step 4. Step 5 introduces only an email notification message and contains no link. Concluding, all the links (except the very first) are provided to the participants before being accessed by them—making the inserted RESTful choreography fragment hyperlink complete.

Figure 40 shows the output of our overall approach. It implements unambiguously the parts procurement decision (modeled in Figure 39) so that all suppliers take the correct path following the exclusive split gateway. The derivation steps are annotated in red.

8.3 CONCLUSIONS

This chapter tackles a problem related to the implementation of choreographies' exclusive gateways. We discuss the limitations of the BPMN specification regarding the implementation of choreography's exclusive gateways and state the problem induced by these limitations. Once the problem has been clearly stated, we suggest a solution by introducing the RESTful decision service—a service that is based on the Decision Model and Notation standard.

The RESTful decision service provides a REST interface to allow the proper interaction of the participants with the decision service. Additionally, we provide a stepwise method for embedding such a service in any choreography diagram. The result is a partial RESTful choreography that describes the complete interaction. The RESTful decision service makes the decision-relevant data visible and consistent assuring that the decision is always executed on the same input data. Using the DMN standard's FEEL expressions eliminates the misinterpretation of the decision logic and, hence, output. Moreover, using DMN we achieve an increasing degree of separation of concerns between decision and process logic, therefore, improving the choreography's comprehensibility and maintainability [5].

Finally, we illustrated our approach with an example which represents the interaction of a single business actor with an undefined number of business partners. Future work will look at more complicated interaction patterns with more complex decisions, where we believe that the RESTful decision service will play even a larger role for facilitating the implementation of decisions in choreographies. The idea presented in this chapter—having a global decision model complementary to a choreography model—has been further investigated in the context of blockchains by Haarmann et al. in [27]. Instead of a RESTful service,

the authors propose a service based on the Ethereum Blockchain[1] to deal with participants that do not necessarily trust one another.

1 https://ethereum.org

9

RESTFUL CHOREOGRAPHY GUIDE

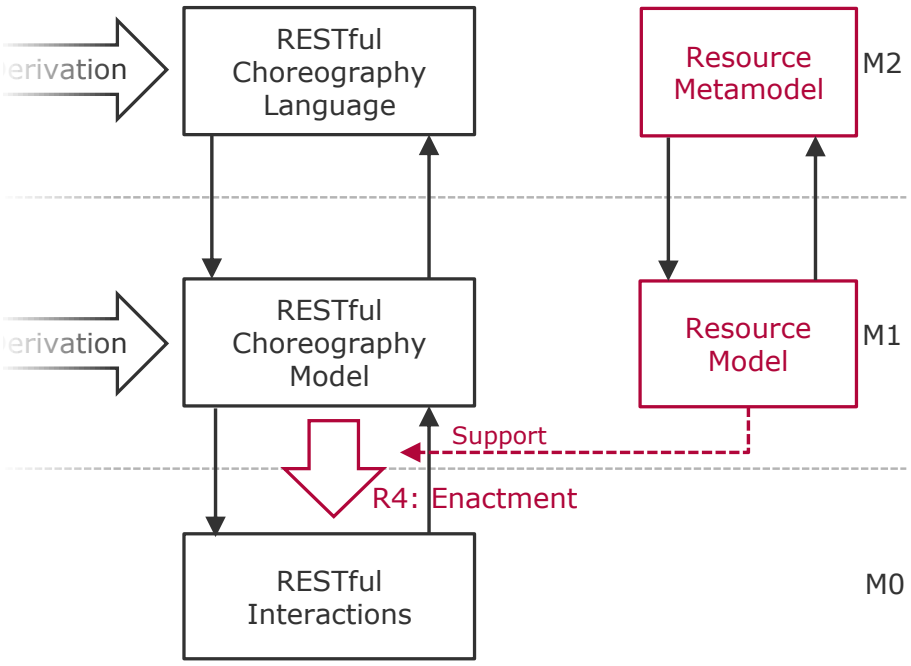

Figure 41: ChoreoGuide

In the previous chapter we proposed a solution for facilitating the implementation of the choreography's exclusive gateway. In this chapter, we address the more general problem of implementing the whole RESTful choreography. As with any choreography, we identify two categories of problems that might arise during the RESTful choreography execution: invalid message, and a wrong order of interactions. Being in a distributed setting, it is the responsibility of the task initiator to send the correct message at the correct order, as it is up to task recipient to check the validity of the message and the order. In this chapter, we leverage the validity checking on the global level by introducing RESTful Choreography Guide (ChoreoGuide) - a central RESTful service acting as an intermediary that guarantees the correct execution of the choreography between the business actors. Like in with the previous contribution, also this chapter's contribution is considered a partial fulfillment of requirement *R4: Facilitating the enactment of RESTful choreographies*.

ChoreoGuide is derived systematically from the RESTful choreography and the Choreography Resource Model - a collectively agreed data model with constraints on the exchange of REST resources. ChoreoGuide per se is a business process model exposed via a REST interface and

can be deployed on a RESTful business process engine to be executable. Hence, the choreography participants do not necessarily require a process engine or a complex system in place to interact with it. We benefit from the separation of concerns, i.e. all participants are clients (from the REST perspective) to the same server (ChoreoGuide). This way the Choreoguide can evolve independently from the clients underlying systems as long as the REST interface is kept unchanged. This is the case when the service can be used by third parties as a starting point to create new business models that bridge the interaction between business actors and clients like Airbnb[1] and Easychair[2]. This work is based on our published paper in [59].

Before we describe in details what ChoreoGuide is, it is very important to state what ChoreoGuide is not. ChoreoGuide is not a full orchestration of a choreography but rather an hybrid between a choreography and orchestration. It is not an orchestration in a service context because, by definition, an orchestrating service actively calls other different services to reach its own goal and the services are, mostly, not aware of that goal [79]. ChoreoGuide is a reactive service, in that it just checks the request payload and answers whether it is correct and timely or not. It is not concerned with the business logic and does not take an active role in the interaction.

9.1 PROBLEM STATEMENT

This section formulates the problems addressed in the rest of the chapter. To illustrate the problem, and consecutively the solution, an example scenario is introduced. Figure 42 describes the interaction of three participants involved in a purchase process. Each of the participants provides its own RESTful API. The *skateboard manufacturer* sends an order to the ball bearings *supplier*. The *supplier* can confirm the order or reject it. In case of the former, the *supplier* sends the ball bearings to the *skateboard manufacturer* as requested in the order. Once the ball bearings are delivered, the *skateboard manufacturer* determines the percentage of defective units over the total purchased amount. If the percentage is lower than 5% the manufacturer initiates a payment request to the *payment organization* (the third participant). If the payment is accepted, the *payment organization* confirms the payment to the *skateboard manufacturer* and notifies the *supplier*, thus successfully concluding the purchase. Otherwise, the *skateboard manufacture* can initiate again a payment or cancel the order. As in the case when the order is canceled due to the high percentage of defective units, the *skateboard manufacturer* sends back the ball bearings, thus, leading to an unsuccessful purchase.

1 https://www.airbnb.com/
2 http://www.easychair.org/

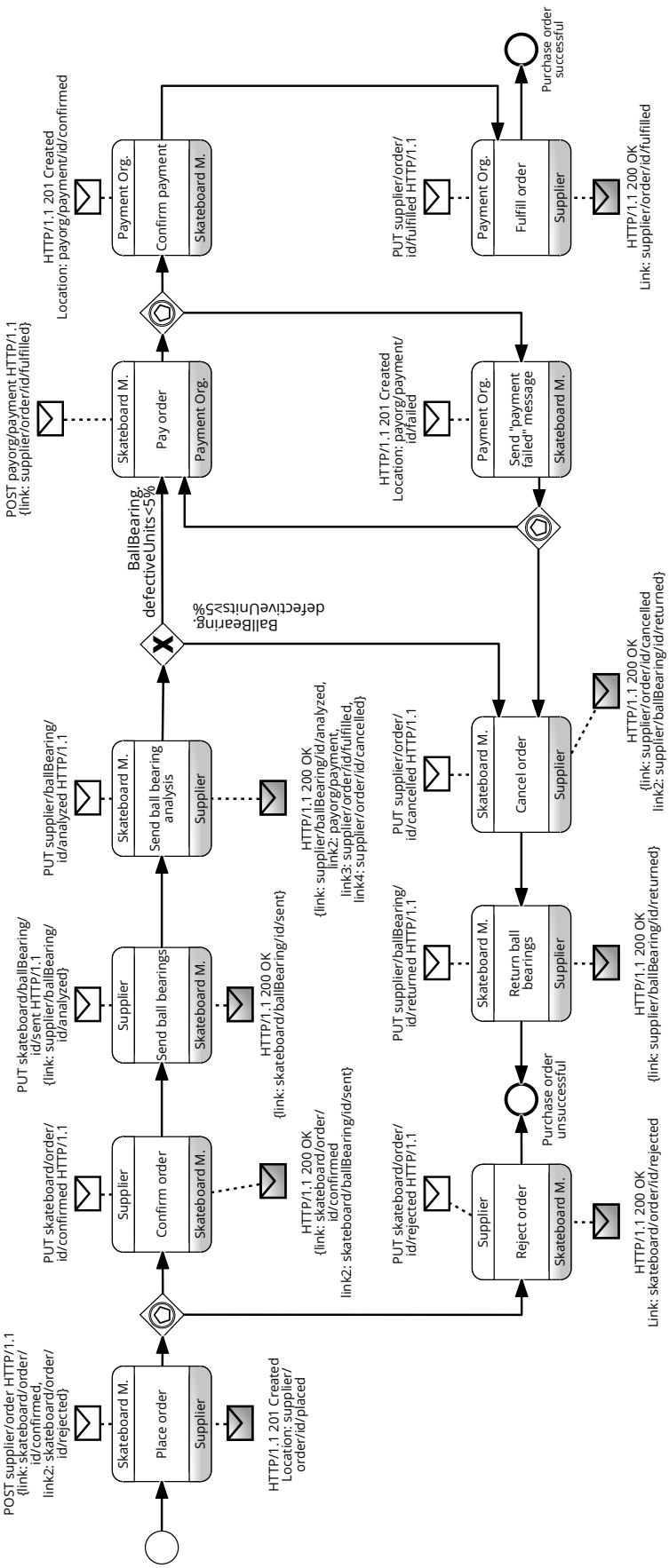

Figure 42: RESTful choreography for the purchase of ball bearings

In general, two things can go wrong during the execution of a choreography: invalid message and wrong message exchange order. In a RESTful choreography that translates to:

1. invalid REST request
2. wrong request order

Considering the first problem, RESTful choreographies that are complete (see Definition 7.2) ensure the correctness of the REST interfaces. Therefore during the execution, if an invalid REST request is encountered, the problem is caused by the request payload and not the REST interface. The following two concepts briefly describe the nature of the payload intricacies encountered in RESTful interactions that are likely to be the cause of invalid request payload.

REST RESOURCE SYNCHRONIZATION When more than one participant provides a REST API, there are REST resources that reside and are managed by different participants concurrently but refer to the same concrete concept, for example *order* and *ball bearings*. This means that every resource state change needs to be propagated to all participants containing and concerned with that particular resource. For example, the *ball bearing* is a REST resource offered by the *supplier* and the *skateboard manufacturer* with the respective resource URIs: *supplier/ballBearing/id* and *skateboard/ballBearing/id*. Both resources represent the same real object—the ball bearings. For every REST request the client has to address the resource lying on the server side and, if the request has been accepted, the client's local resource needs to be matched. For example, the *supplier* sends the request *PUT skateboard/ballBearing/id/sent HTTP/1.1* and has to make sure that its local *supplier/ballBearing/id* resource is synchronized with the state *sent* as well. In addition, hyperlink completeness and correct resource behavior become complicated to check because there are many internal changes that are not reflected in the RESTful choreography.

RESOURCE REPRESENTATION MATCHING In REST, resources are manipulated through their representations. For a client to interact with the resource, the resource representation has to be retrieved, manipulated, and, then, submitted to the server. In a setting where the choreography's participant roles can be instantiated by a plethora of diverse business actors, the resource representations have to be interpreted and used correctly, as well as be matched with the local resource ad hoc for every pair of participants.

Despite service matching being a well-researched topic in the area of SOA [26], we propose a REST-based lightweight solution that provides a single view on the involved resources. This allows the prospective business actors to adapt their own local resources to a single global resource representation model. Hence, the possible payload problem

caused by the *resource representation matching* is mitigated, i.e., all participants have a resource reference model to match against. Moreover, if the states of the dynamic resource instances are managed by a global entity, the possible payload problem caused by *resource synchronization* is eliminated, i.e., the participant can retrieve the up-to-date resource state at any point during the choreography execution.

The second main problem—wrong request order—can be caused by the initiators that sends their request in an order that is not prescribed by the choreography. This can be particularly hard to detect from the perspective of a recipient that is not involved in the previous interaction. Therefore, having a choreography global state (accessible by the participants) resolves the problem of enforcing the RESTful choreography's control flow.

Moreover, due to client-server REST constraint, the participants have to switch between the client (sending the request) and server role (sending the response). We argue that the continued role switching harms the separation of concerns, which ultimately impacts the understandability, development, maintenance and evolution of each participant's RESTful service.

As it can be already inferred, our solution revolves around a third party entity that aides the RESTful choreography execution. Specifically, we propose a novel approach towards implementing choreographies by introducing RESTful Choreography Guide (ChoreoGuide)- a central RESTful service that guides the participants along the choreography and assures the proper exchange of resources between them. This service acts as an intermediary between participants making sure that the participants adhere to the choreography they agreed upon and are sending valid requests to each other. All the participants are REST clients who interact with each other through a central RESTful service and, hence, are not required to provide RESTful APIs.

9.2 RESTFUL CHOREOGRAPHY GUIDE

In this section we describe ChoreoGuide as a blueprint for developing RESTful services that guide choreographies. Since the approach revolves around checking the order of message exchanges, i.e., their causal dependencies, we specify ChoreoGuide as a BPMN 2.0 process model enriched with REST-specific information that can be deployed in any RESTful business process engine. A technology-agnostic approach is purposely chosen so that it can be adopted and applied to different technologies. Nevertheless, we discuss an implementation architecture for deploying ChoreoGuide onto a concrete platform (see Section 9.3).

Given that the two main problems during the RESTful choreography execution are the invalid REST request and the wrong request order, the ChoreoGuide has two functional requirements:

1. Validate the REST request payload

2. Check the control flow of the request message exchanges

In a RESTful choreography, the REST requests contains a REST verb, the resource and its state, e.g., *PUT order/id/confirmed*. In order to check the validity of the resource and its state we need more information about the data being exchanged than provided by the choreography. For that, the participants need to agree on a common choreography resource model (see Section 9.2.1). This model is required by ChoreoGuide to check the validity of the resource and its state. The choreography resource model is specified using two standards: UML class diagram and Object Constraint Language (OCL) [64].

Regarding the second requirement, the control flow and the message exchanges in a RESTful choreography are modeled from a global perspective. We provide a mapping, detailed in Section 9.2.2, of the choreography control flow to business process control flow where message exchanges are implemented by process activities.

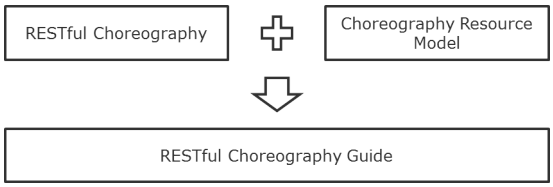

Figure 43: Approach overview

Figure 43 provides the overview of our approach for implementing the RESTful choreography via ChoreoGuide. The approach takes as input a RESTful choreography and its resource model. The target model is a BPMN [65] process model with embedded REST information that can be deployed to a business process engine capable of providing a RESTful API.

9.2.1 *Choreography Resource Model*

In this section, we describe the choreography resource model. The resource model is represented in part as an UML class diagram (see Figure 44) that serves the purpose of specifying the main REST resources, their state and their relation to each other and to the choreography participants as well. It is highly influenced by the domain area and specified by the involved participants alongside the RESTful choreography diagram at design time.

Figure 44 illustrates the static view of the choreography resource model for our running example modeled in Eclipse Modeling Framework[3]. Every *ResourceModel*, independently of the use case, is always composed of at least two *Participants* and at least one *RestResource*. Depending on the use case, participants and REST resources are added as classes that inherit the abstract classes *Participant* and *RestResource*

3 https://eclipse.org/modeling/emf/

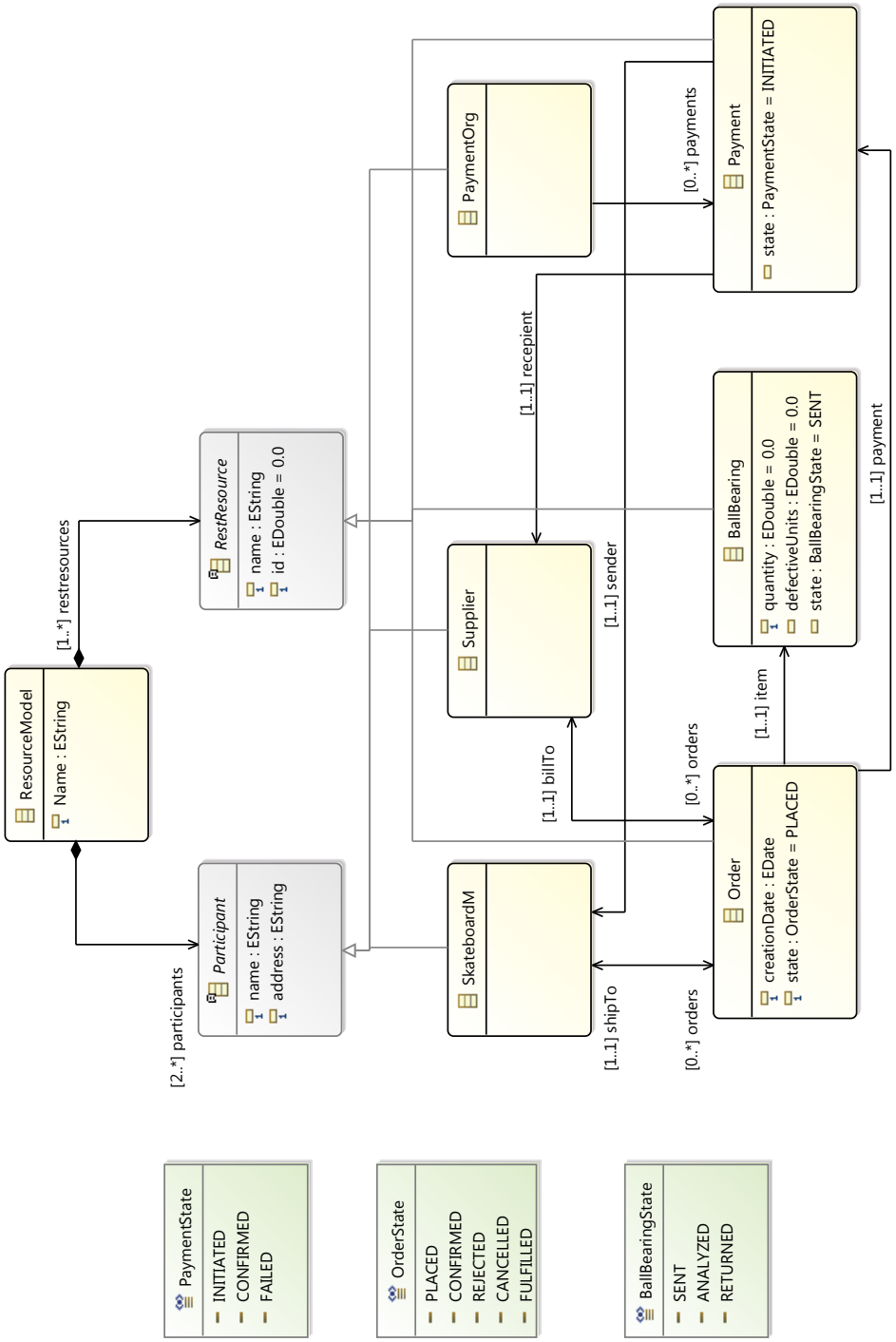

Figure 44: Choreography Resource Model (static model)

respectively. The attributes and the associations are specified according to the use case domain. For example, *Order* has a *creationDate* and *BallBearing* as an *item*.

However, a static model of the resources is not sufficient. Resource states are not always mere attributes that can be directly assigned by participants. The state represent certain conditions of the resource which is aggregated from the concrete value assignment to the resource attributes. For example, resource *order* is in the state *placed* only if the quantity of the ball bearings is greater than 0. We use OCL [64] to formally express such conditions that must hold during the entire execution of the choreography. Below is the respective OCL expression for the aforementioned condition.

```
context order
inv nonEmptyOrder: self.state = OrderState : : PLACED
implies self.item.quantity > 0
```

The OCL expression refers to the UML class diagram. In this expression we look at the class *Order*. Then we identify an invariant (*nonEmptyOrder*) that must hold during the instantiation of the *order* objects. The invariant in this case is an implication where the premise is the resource in state *PLACED* and the consequence is an expression that must hold for that state. In fact, this is the general pattern we propose to check the validity of the state for any resource object:

```
context <<resource>>
inv <<resource state>>
implies <<condition>>
```

In order to use the OCL expression, each *RESTResource* child class must have an attribute *state* of the type *Enumeration* that contains all the possible states of that particular resource, e.g., *Payment* can be in state initiated, confirmed, or failed. The condition of resource state can be arbitrary complex and long (to the extend allowed by OCL). OCL provides the possibility to navigate the class diagrams and express complex relations between resources. Using a standard like OCL allows the developers to choose the desired tools that support the evaluation of OCL expressions. Another key benefit is that OCL is free from side effects. This means that checking whether an OCL expression holds does not affect the running program. When it comes to choreographies, this is very useful because ChoreoGuide should only reply whether the state change request is valid or not without interfering with the internal logic of the participants' process.

9.2.2 *From RESTful Choreography to Process Model*

In this section we present how the process model is systematically derived from the RESTful choreography and the resource model. We consider the translation of the two type of RESTful choreography's tasks (REST tasks and email tasks) as well as three types of gateways: parallel gateway, data-based exclusive gateway and event-based gateway as the

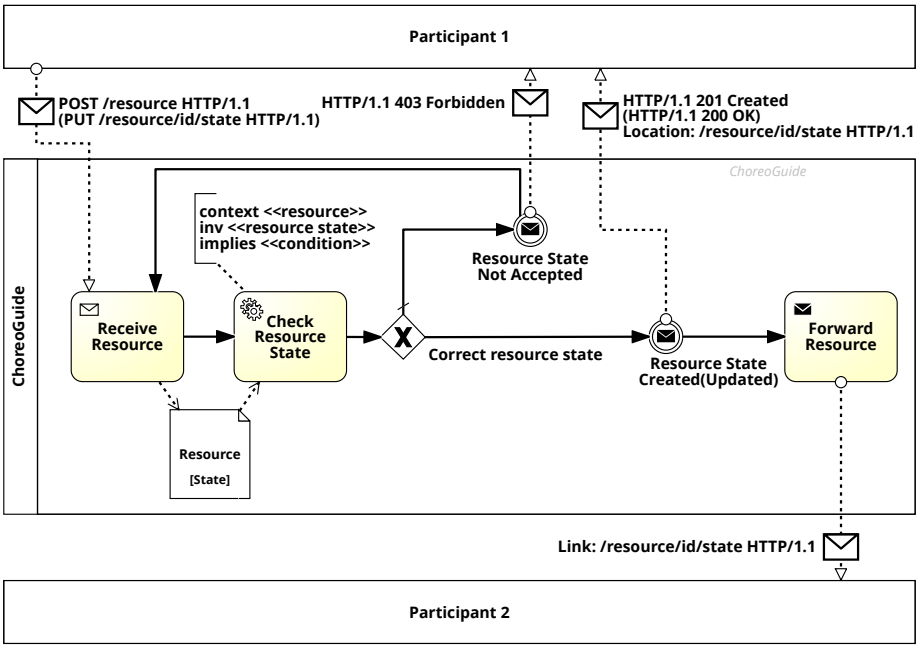

Figure 45: Business process construct for *POST* and *PUT* (in brackets) REST tasks

most commonly used choreography elements. In the following, these choreography elements are mapped to business process constructs. After that, all the individual derived constructs are concatenated to form the whole target business process.

In a RESTful choreography, there are four types of REST Tasks: *GET*; *DELETE*; *POST*; and *PUT*. The latter two represent a resource creation or a resource state change. These two kinds of tasks are translated to the following business process constructs (see Figure 45): a receive task; a service task; an exclusive gateway which branches the control flow in two parts; a "fail" message intermediate throwing event; a "successful" message intermediate throwing event; and a send task that forwards the initiator's request. Specifically, as illustrated in Figure 45, the receiving task is accessed from the participants by the REST request specified in the REST task, e.g., *POST /order/ HTTP/1.1*. In case of a *POST* or *PUT* a resource or a representation of it is delivered as a message payload to the server. With *POST*, a new resource is created in an initial state and *PUT* is used to update a resource with a new state. After the resource is received from the server, an automatic check is performed in the subsequent step by a service task. This task automatically checks whether or not the state-change request is valid by evaluating the respective OCL expression in the resource model. In case the OCL expression is evaluated to false, the server replies to the client (the participant who sent the request) with a *HTTP/1.1 403 Forbidden* status code. This status codes is reserved for cases where the server understands the request but refuses to process it and it replies with a payload that explains the

problem[4]. Otherwise, the server replies with a *HTTP/1.1 201 Created* (for *POST*) or *HTTP/1.1 200 OK* (for *PUT*). RESTful choreography diagrams model only the successful interaction between the client and the server. Hence, the response of a valid request used in the process model is derived from the choreography. Finally, the link for getting the new resource state is forwarded to the recipient.

GET is used to read the state of the resource and *DELETE* to remove the resource from the location specified in the URI. In case of *GET* task, the state of the resource does not change and, therefore, there are no conditions to be checked. The corresponding process model (see Figure 46) has three consecutive nodes connected by sequence flows: a receiving task for the request, a message intermediate throwing event for replying to the initiator by sending the resource; and, a send message task for notifying the recipient that the resource has been read.

When a *DELETE* request is sent, there are no state conditions to be checked (similarly to *GET* task) in terms of attribute values. Hence, the corresponding process orchestration is the same as in the *GET* case (see Figure 46), but a "resource deleted" message notification is sent back in lieu of a reply containing the resource. It is worth mentioning that the delete request is not forwarded to the participant for permission prior to the server reply because the delete request is part of the intended behavior specified by the RESTful choreography. If any participant would arbitrary request to delete a resource, ChoreoGuide will not accept the request unless it is part of the designed behavior.

Choreography's parallel gateway with m outgoing sequence flows is mapped to the following business process construct: a send task that sends a message containing n links where n is the number of choreography tasks that immediately follow the parallel gateway. The send task is followed by a process parallel gateway (same syntax with the choreography's parallel gateway) with the m outgoing sequence flows. n equals m when the choreography parallel gateway is immediately followed by only REST tasks.

Similarly to the parallel gateway, the choreography's event-based gateway with m outgoing sequence flows is mapped to a send task followed by a process event-based gateway with m outgoing sequence flows. The send task sends a message containing n links, where n is the number of REST tasks that immediately follow the gateway. A concrete example is depicted in Figure 47.

The choreography's exclusive gateway follows the exact mapping as event-based gateway - the target process construct consists of a send task followed by process exclusive gateway (see Figure 46). However in this case, the outgoing sequence flows are conditional sequence flows which also need to be mapped accordingly. It means that the conditions need to be evaluated from ChoreoGuide. As mentioned in the previous

4 If, due to security policies, the reason has be opaque, an *HTTP/1.1 404 Not found* response code can be used instead

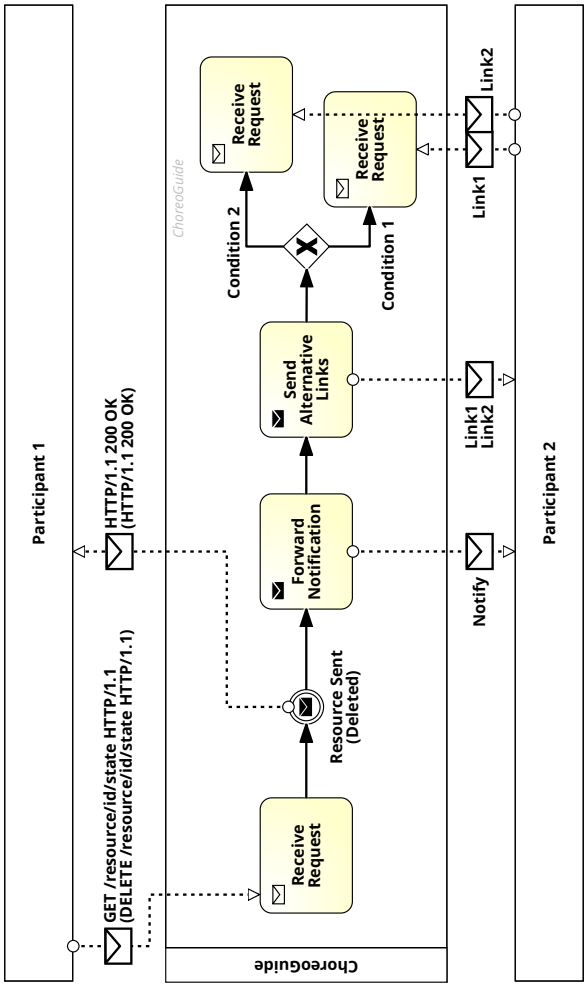

Figure 46: Business process construct for *GET* and *DELETE* (in brackets) REST tasks followed by the construct for exclusive gateway

chapter (see section 8.1.1), the data upon which the conditions are evaluated is passed previously to the participants affected by the gateway (participants who immediately follow the gateway). For example, the number of defective units is passed from the skateboard manufacturer to the supplier via the ball bearings analysis. This value is found in the instance of the resource model (see Figure 44). Since ChoreoGuide manages the resource model, it has sufficient data to evaluate the conditional flows. This constitutes a lightweight solution (compared to the elaborated solution proposed in the previous chapter) that solves the problem of implementing the choreography exclusive gateway. Nevertheless, a RESTful decision service may be integrated into the ChoreoGuide for executing complex decisions.

Email tasks serve two purposes in a RESTful choreography: send an hyperlink to the recipient for addressing certain resources; and, inform the recipient about a resource state change. In this sense, email tasks are not considered in this mapping because they are not needed in the original RESTful choreography model when all interactions go through the

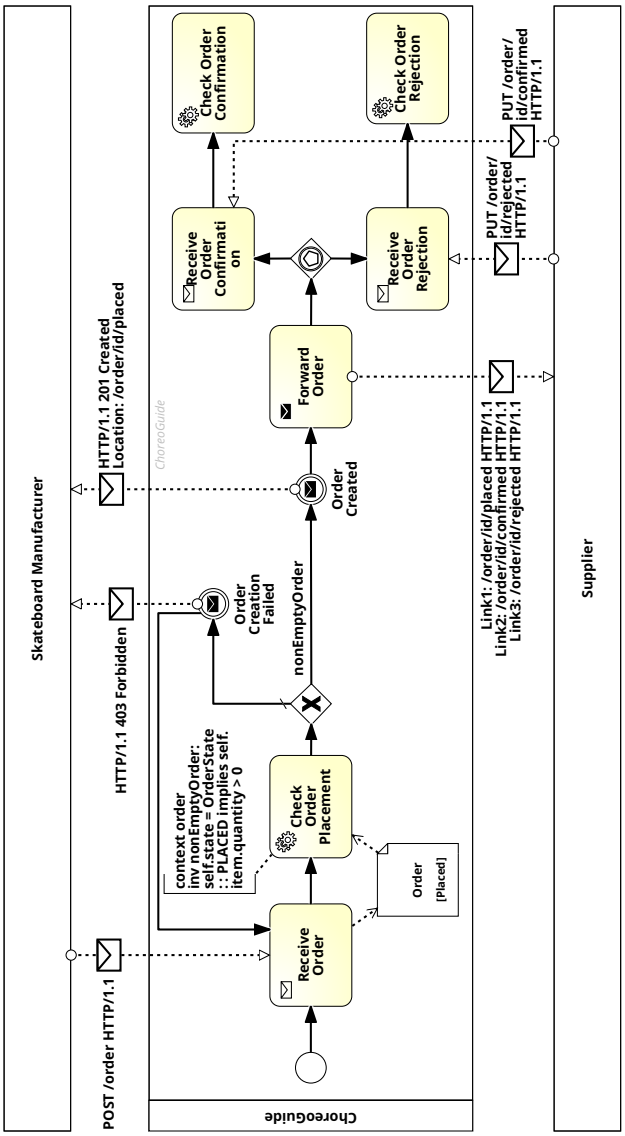

Figure 47: ChoreoGuide excerpt sample of the running example

ChoreoGuide. A participant does not need to inform directly another participant by email because the ChoreoGuide can forward the hyperlink or the notification that a certain REST resource has been changed.

Finally, the choreography start events, end events and join gateways are mapped accordingly to ChoreoGuide's business process. The concatenation of all generated business process constructs is realized by connecting them with a single sequence flow as defined in the RESTful choreography. After the concatenation, a single reduction rule is applied: For every gateway in the RESTful choreography that immediately follows a REST task, the Forward Resource/Notification task is merged with the send task preceding the gateway (the task sending the links) if the message recipient is the same (like in Figure 46). The payloads of the outgoing messages (of the send tasks before the merge) are added up and form the new message's payload. A snippet of the resulting

mapping from our running example is depicted in Figure 47. The Forward Order send task is merged with the send task that immediately precedes the event-based gateway. The resulting task, besides notifying the supplier about the order placement, sends the links needed to execute the upcoming event-based gateway.

9.3 IMPLEMENTATION ARCHITECTURE

This section presents the architecture of the proposed approach and discusses how the generated ChoreoGuide process can be deployed to an existing process engine. Figure 48 shows the main components of the architecture, 1) the parser responsible for parsing the RESTful choreography and resource model, 2) the generator that derives the ChoreoGuide orchestration process and deploys it, and 3) the process engine Chimera[5]. Chimera is an academic process engine, developed to validate research approaches in the field of BPMN and case management. It is a good fit for RESTful choreographies, because it exposes running processes via a RESTful API.

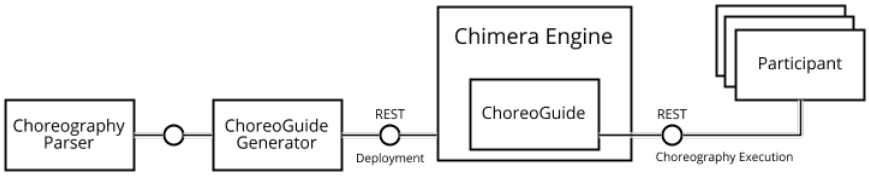

Figure 48: Architecture of ChoreoGuide

The Chimera engine requires process model to define the data classes it operates on by providing a domain data model. For ChoreoGuide we use the sub-classes of *RESTResource* class—the concrete resources. The choreography resource model is used for defining the data objects and their attributes. The data objects represent concrete resources, like *order* and *ball bearings*, and their state. Below we discuss the peculiarities of deploying ChoreoGuide into Chimera.

URI MAPPING. As part of the mapping to a concrete execution engine the addressing of ChoreoGuide and participants has to be considered. Since participants will conduct a choreography multiple times, the engine needs to ensure that messages are correlated to the correct *instances*. Additionally, the Chimera RESTful API makes data objects available only in the context of a process instance. Therefore the URI annotations used in the RESTful choreography, e.g. */resource*, need to be prefixed by the following parts: 1) base URI where the engine is reachable in the network, 2) identifier of the choreography, 3) identifier of the choreography instance. The resulting URI for an order resource,

5 https://github.com/bptlab/chimera

thus, might look like *https://example.org/chimera/api/choreography/5/instance/23/order*. In case that one choreography instance handles multiple resources of the same type, e.g., multiple orders, an identifier of the concrete order needs to be appended to the URI.

NOTIFICATIONS. Notifying the participant for any change in the ChoreoGuide execution state can be achieved in different ways. Chimera provides two methods: emails and service calls (when the participant provide their own RESTful API). These have to be configured during the deployment of ChoreoGuide on the Chimera engine. A simple email suffices as vessel for hyperlinks. In our example the supplier receives an email with a short order description and two hyperlinks (see Figure 47) to either reject or confirm the order. Depending on the systems used by the participants other push technologies could be used as well [77].

INSTANTIATING. When the initiator of a the first choreography task sends the initial message no instance of the ChoreoGuide orchestration exists yet. Therefore, the first message needs to be translated to a *POST* to the Chimera engine for instantiating the ChoreoGuide instance.

RESOURCE STATE VALIDATION. At last, we discuss how the OCL conditions are evaluated in the process engine. The OLC conditions refer to attributes of resources defined in the resource model, which corresponds to the data objects managed by the ChoreoGuide instance. Hence, the conditions are used as annotations on the sequence flows following the exclusive gateway that reflects the correctness of the resource state.

9.4 CONCLUSIONS

In this chapter we introduce a novel approach towards implementing business process choreographies. Existing work, focus on deriving public processes for each participant and enforce their execution. We propose a central RESTful service that takes the participants "by the hand" and guides them through every step along the choreography. The service employs the REST architectural style to take advantage of the REST constraints.

One key benefit of our approach is that the participants are not required to run a process engine or a complex system to interact with each other. This is particularly important for the interaction with human clients that can participate in the choreography via a simple Web browser. Our approach fully employs the principle of HATEOAS, i.e hyperlinks are indeed the engine of the choreography state.

Another important benefit is that ChoreoGuide not only checks the control flow but validates the correctness of the resource state changes using the choreography resource model. In addition, having such a

model helps the participant to agree on a common data structure and avoid problems coming from the misinterpretation of the inter-organization data.

Part IV

EVALUATION AND CONCLUSIONS

10

EVALUATION OF RESTFUL CHOREOGRAPHY DERIVATION

This chapter evaluates the derivation approach (Chapter 6) that takes as input a business process model and outputs a RESTful choreography. First, we explain the architecture of our prototypical implementation. Then, we present the results on the accuracy of the derivation steps for a set of 172 choreography diagrams from practice. We conclude with a discussion over the evaluation results. The results of these evaluation are published in [60].

10.1 REST ANNOTATOR IMPLEMENTATION

For evaluating our approach, we developed a tool, called *REST Annotator*. Its architecture is depicted in Figure 49 as an FMC diagram [33]. The *REST Annotator* takes a set of business process choreography diagrams as an input and it outputs a set of RESTful choreography models. The tool makes use of three external components: the *Label Annotator* by Leopold et al. [39], *WordNet* [53], and the distributional similarity component of the *DISCO tool* [34]. The main component that constitutes the tool is composed of three sub-components: *Label Analyzer*, *REST Verb Identifier* and *REST Task Generator*.

The *Label Analyzer* is responsible for extracting all the labels from the model and analyzing them with the help of the *Label Annotator*. The latter is used to notate the action and the business object of a choreography task label. The *Label Analyzer* provides the action and the business object for each label to the *REST Verb Identifier* and the *REST Task Generator* components. The *REST Verb Identifier* component requires the action provided by the *Label Analyzer* and the synonyms of *WordNet* resembling the respective REST verb. If no synonym is found, the component requires the semantic similarity score between the action and the synonym sets of the REST verbs from the *Disco Semantic Similarity* component. Once the semantic relation of the action with each of the REST verbs is identified, the REST verb and its respective score is passed to the *REST Task Generator* component. This component has enough information to generate the REST request and hence the REST task. The final output of *REST Annotator* the set of RESTful choreographies.

Additionally, the Label Analyzer, before providing the final action and business object, needs to apply the advanced derivation approach described in section Section 6.4. To this end, it requires $\mathtt{Syn}_{\mathtt{Send}}$, which is generated by the *REST Verb Identifier*. Moreover, the latter needs

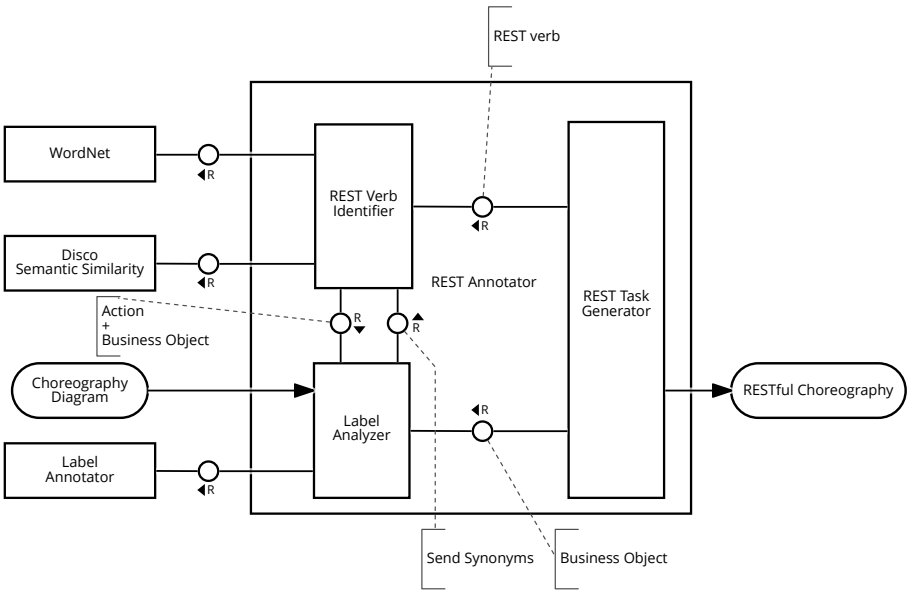

Figure 49: The REST Annotator architecture

from the former the business object to check for the presence of an indefinite article. As explained in subsection 6.4.2, the presence of an indefinite article can alter the output of the *REST Verb Identifier* from *PUT* to *POST*.

10.2 EVALUATION SETUP

As evaluation data, we use choreography diagrams from the BPM Academic Initiative[1]. The initiative offers a rich set of process models from different domains. Overall, we retrieve 424 BPMN choreography diagrams. Since these diagrams are created from experts and non experts alike, it is necessary to *clean the data*. We apply the following cleaning criteria:

1. *English-only Diagrams*. We include only diagrams with English text labels. This criteria is necessary because our approach relies on natural language analysis components that only support English.

2. *Syntactically correct Diagrams*. Diagrams which have syntax errors with respect to the BPMN 2.0 choreography diagram specification are excluded.

With regard to the *evaluation procedure* a REST expert had to perform a three-step evaluation for each choreography task: a) the syntax correctness of a label, b) the adequate generation of the REST verb, and c) the suitability of the generated REST URI. In case a) holds true, the

1 https://www.signavio.com/bpm-academic-initiative/

evaluator further has to check if the identified REST verb with the best matching score is adequate. In case b) holds, the evaluator has to check if the the generated URI is suitable.

The evaluation comes, however, with its own limitations. The existence of a single evaluator may impose subjectivity in determining the correct match. To mitigate this issue, the evaluator repeated several time the inspection procedure while consulting common practices in developing RESTful APIs. This avoids errors like inconsistent verification for the same label - RESTful request pairs and helps to resolve the non-obvious pairs.

10.3 EVALUATION RESULTS

This section discusses the results which are summarized in Table 6. The 172 models contain 1213 choreography task labels in total. The models size (in terms of choreography task count) ranges from 1 to 26 with an average of about 7 choreography task per model. From these labels, 864 labels (71.14%) are syntactically correct labels. In the following discussion, we only focus on those labels that are syntactically correct and discuss how the verb identification and the link generation performs in these cases.

Table 6: Quantitative Results of the User Evaluation

Total No. of Labels	1213
No. of syntactically-correct Labels	864 (71.46%)
No. of syntactically-incorrect Labels	349
Total No. of Correctly Identified REST Verbs	772 (89.35%)
.. with the Synonym Identification Strategy	322
.. with the Similarity Identification Strategy	450
.. POST	139 (97,89%)
.. PUT	577 (88,36%)
.. GET	54 (80,60%)
.. DELETE	2 (100,00%)
Total No. of Incorrectly Identified REST Verbs	92 (10.65%)
Total No. of Correct Request URIs	723 (93.65%)
Total No. of Incorrect Request URIs	49 (6.35%)

The *verb identification strategies* have identified the correct REST verb in 772 labels which amounts to 89,35% of all syntactically-correct labels. Among these labels, we further distinguish between the verbs that have been identified with the synonym strategy and the similarity strategy. The synonym strategy is capable to derive the correct REST verb in 332 labels, while the similarity strategy derives the correct REST verb for 450 choreography labels. The results emphasize the need for the similarity identification strategy of the REST verb. The most identified REST verb is PUT as it is expected in a choreography context where participants change the state of business objects, e.g., order is sent, accepted, delivered and payed. POST was identified in 139 cases, 8 of which were identified using the presence of the indefinite article. The low number of DELETE identifications reflects also the rare cases of using DELETE in the REST context due to resources being often archived or saved in a particular state rather than being deliberately deleted.

In total, 92 choreography labels (10.65%) have been annotated with the wrong REST verb. We observe that GET is detected the least and DELETE is always detected. We identify two classes of errors that can lead to the wrong annotation, the first of which is fixed in the context of this sample set and does not count towards the incorrect REST verb identifications. This first class subsumes choreography labels for which the similarity strategy revealed two or more equal similarity scores. This has been the case for 101 choreography labels. After identifying the list of these REST-ambiguous actions for this particular sample set, a REST expert was asked to choose the most appropriate mapping. The following non exhaustive list is disambiguated: {*start-PUT, pay-PUT, invoice-PUT, article-PUT, enter-PUT, publish-PUT, allocate-PUT, explain-PUT, disburse-PUT, receipt-PUT, show-GET, book-PUT*}. This list can be used and enriched further with verbs that score equally in the semantic similarity approach.

The second class covers such cases in which our approach identified the wrong verb. The REST evaluation has revealed 92 choreography labels for which our approach did not find the correct REST verb. These cases have to be corrected by the user.

The approach for generating the request URIs has created 723 correct and 49 incorrect URIs out of 772 correct verb identifications. We identified the labeling quality as a main cause for the incorrect URIs. For example, we found choreography tasks that have not been specified correctly by referring to a particular state, e.g. *payment confirmed* and *invoice sent* are mapped to *PUT /payment/id/confirmeded HTTP/1.1* and *PUT /invoice/id/sented HTTP/1.1* respectively. A correct result is generated for the labels *confirm payment / payment confirmation* and *send invoice*. Another cause for the incorrect link generations is the misidentification of the business object. For example, the label *ship article* is labeled as α(ship article) = *article* (action) and β(ship article) = *ship* (business object). The correct labeling would be to identify *ship* as the action and

article as the business object. Nevertheless, we conclude that the URI generation works satisfactory and produces a large number of correct REST URIs.

Table 7: REST request results for the corresponding RESTful tasks from Figure 26

RESTful Task	**REST Request**
create CFP	POST /cfp HTTP/1.1
publish CFP	PUT /cfp/id/published HTTP/1.1
submit a paper	POST /paper HTTP/1.1
start review process	PUT /reviewProcess/id/started HTTP/1.1
assign paper review	PUT /paperReview/id/assigned HTTP/1.1
send review request	POST /review HTTP/1.1
enter paper review	PUT /paper/id/reviewed HTTP/1.1
finish review process	PUT /reviewProcess/id/finished HTTP/1.1
submit paper decision	PUT /paper/id/decided HTTP/1.1
notify paper rejection	PUT /paper/id/rejected HTTP/1.1
notify short paper acceptance	PUT /shortPaper/id/accepted HTTP/1.1
cancel short paper	DELETE /shortPaper/id HTTP/1.1
confirm short paper	PUT /shortPaper/id/confirmed HTTP/1.1
camera-ready paper submission	PUT /cameraReady/id/papered HTTP/1.1
conference registration	GET /conference HTTP/1.1
confirm paper publication	PUT /paperPublication/id/confirmed HTTP/1.1

We also exemplify the results of our evaluation by applying our approach to the exemplary choreography diagram from Figure 26. Table 7 shows the generated REST requests for the respective choreography tasks. It is assumed for this purpose that in the RMS example all participants are RESTful services, i.e., they interact with each other by sending REST calls. In a simple browser settings, the organizer, reviewer and author are users of RMS. In this example, we assume that they also provide a RESTful API. Consider these services as RMS mobile applications where RMS can push notifications [77] depending on the user role, e.g, notifying the reviewers about the papers assigned for review, sending the paper decision to the authors, informing the organizer when all reviews are submitted.

Figure 50 is a mock-up that depicts an excerpt of the RESTful choreography model generated by applying our approach to the running example. In this figure we show how the REST engineer can interact with the generated RESTful choreography. In this case, the REST engineer is provided with all four generated REST requests (one for each REST verb) ranked based on the matchmaking score (1 being the best and 0 the worse). Depending on the selection the HTTP response is generated automatically, assuming that the interaction is always valid.

10.4 DISCUSSION

Two main observations emerge from the quantitative evaluation results. The first observation relates to the correct annotation of choreography tasks with REST URIs. For example, it identifies *PUT* to be the correct REST verb for the task *confirm short paper* and generates the URI *PUT /shortPaper/id/confirmed*. However, we also encounter problems for cases, in which the approach retrieves several possibilities for REST verbs and fails to make a decision for one particular REST verb. In the example, the choreography task *enter paper review* falls into this group. The approach identifies the REST verbs *PUT* and *GET* because the action *to enter* is not a member of any REST verb synonym list and the semantic similarity score is equal for both REST verbs. Based on this result, the link generator component creates two possible links, among which the user has to choose. Nevertheless, the links themselves have been created correctly. As mentioned in the previous section, we solved this problem for this particular test set by disambiguating the REST verb mapping. However, the list of disambiguated verbs is not exhaustive at its current form because there are other verbs which were not part of the labels we used in our evaluation. The list can be used as input when applying our approach to achieve better results for choreography labels which contain such verbs.

The second observation covers REST requests that are incorrect and that need to be manually corrected by the user. As an example, consider the choreography task *conference registration*, for which our approach

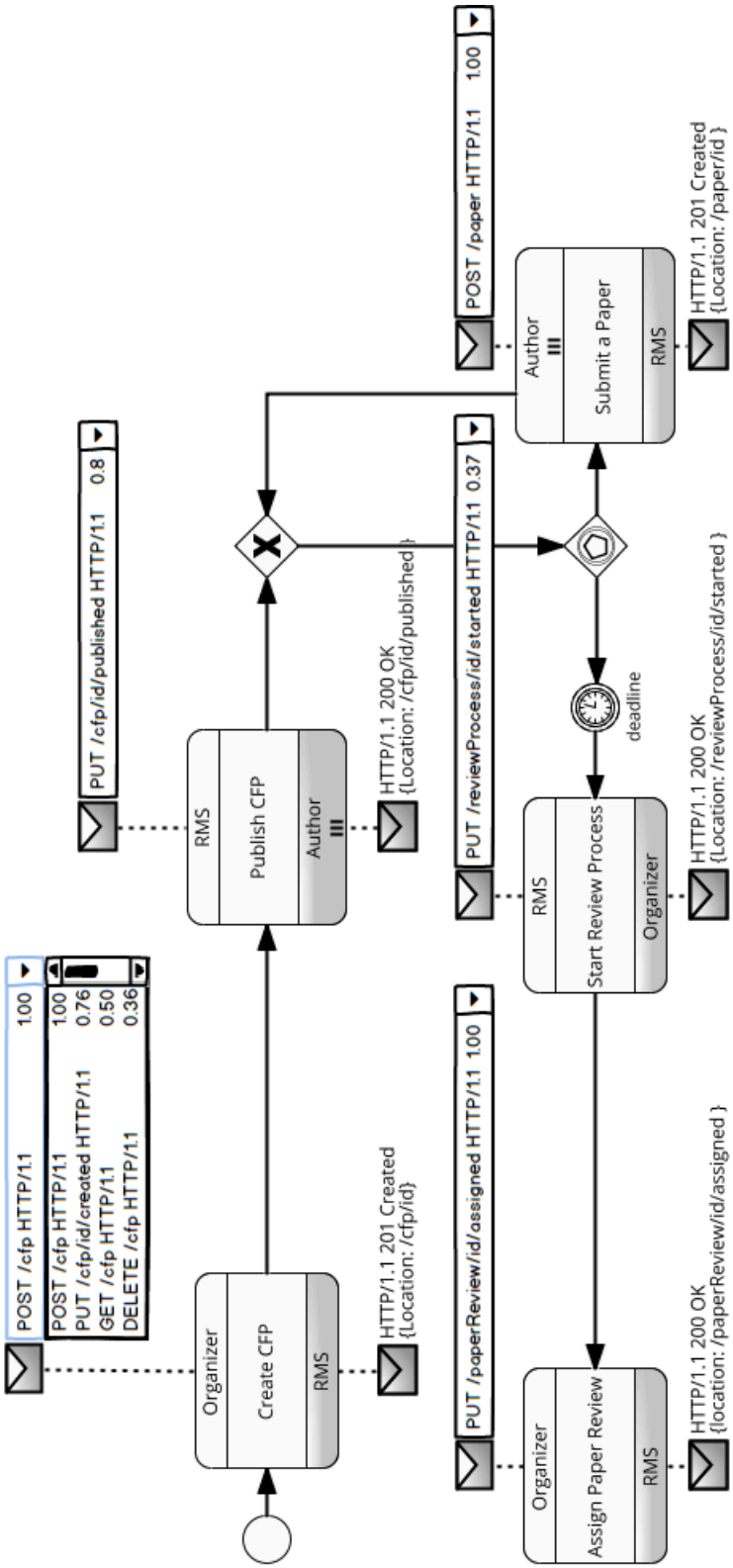

Figure 50: A part of the generated RESTful Choreography of RMS

creates a *GET* link. However, we would expect a *POST* or a *PUT* request. Incorrect links of this type may have several error sources. On the one hand, the Label Annotator component (see Figure 49) might have misclassified the choreography task and erroneously changed action and business object. On the other hand, the REST Verb Identification component might have caused the error because the action is either a direct member of the synonym word lists or its similarity score with the synonym words is highest for one of the other REST verbs. In our example, the former applies. The REST verb *GET* has been identified, since the action *to register* is a WordNet synonym of *to read* and thus a member of the synonym word set Syn_{GET}. Hence, the other alternatives are not considered so far, which finally requires the user to correct this REST request.

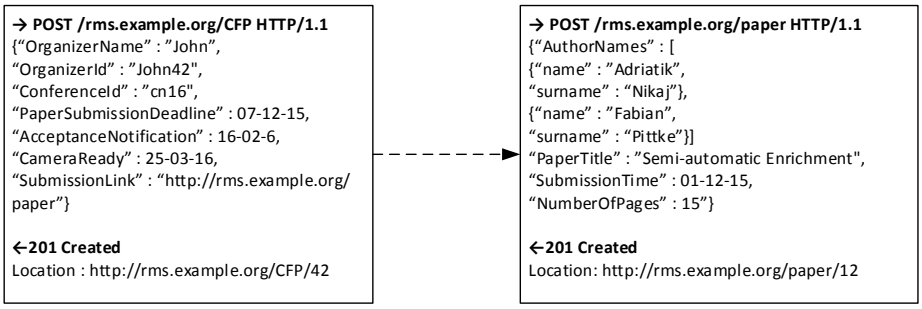

Figure 51: A concrete skeleton instance of RMS implementation

At last, Figure 51 depicts a concrete instance of the RMS RESTful interaction. The part in bold and the order of REST interactions are generated by the REST Annotator tool and provided to the developer as a skeleton to follow for developing the RESTful API. In the RSM context, the two rectangles represent respectively the concrete instances of the *create CFP* and *submit a paper* choreography tasks from Figure 50. The dashed arrow expresses that the second instance can only be executed only after the first one is executed. For a given RESTful choreography, a skeleton diagram can be derived for each participant who offer a RESTful API, like the RMS mobile app of the conference organizer which receives notification from RMS about the status of the reviewing process . Hence, we jump from a global choreography view, to at least one orchestration view that focuses only on the REST behavioral interface i.e. the order in which the REST requests and responses are performed within a single participant application. The benefit of applying our approach is in that the same URI generation logic is used across all participants contributing to a better understandability, maintenance and evolution of REST APIs [74]. The automation of deriving skeletons from a RESTful choreography is left as a future work.

11 CONCLUSIONS

This chapter concludes this thesis by summing up its main results in the first section. The second section discusses the limitations and provides an outlook of future work.

11.1 SUMMARY

In this thesis, we motivate and argue about the need and relevancy of enacting business process choreographies in a RESTful setting (in chapters 1, 2, and 3). To this end, the research questions are formulated in Chapter 4, from which a set of high level requirements are derived. These requirements are addressed in the main chapters by several research contributions, which are summarized as follows:

- *RESTful choreographies.* The RESTful choreography modeling language is designed to capture RESTful interactions into models. Serving as a bridge between business process choreographies and RESTful interactions, it is designed by extending the BPMN 2.0 process choreography language. RESTful choreographies represent refined contracts between business actors at a level of detail that is suitable for implementing REST-specific interactions. Chapter 5 provides a metamodel together with formal properties for specifying RESTful choreographies. Moreover, derivation guidelines and design patterns are introduced to support their modeling.

- *Semi-automatic derivation.* To preserve the separation of concerns between business process choreography designers and Web engineers, a semi-automatic derivation approach is introduced in Chapter 6. Natural language analysis techniques are used to extract domain specific information from process choreography models and use this information to specify REST interfaces among business partners. The approach is estimated to generate correct REST tasks in about 90% of the cases. In the remaining cases, Web engineers have to manually choose the best match. Moreover, having a unified derivation method for generating each participant's RESTful API simplifies the API's understanding and adoption by the involved choreography's participants.

- *Completeness properties.* Formal completeness properties, introduced in Chapter 7, guarantee that RESTful choreographies derived from enforceable and deadlock free process choreographies preserver these qualities. The properties are *hyperlink completeness*

and *correct resource behavior*. They capture the REST-specif behavior induced by the REST information. In addition, we provide a formal analysis solution for checking hyperlink completeness by translating RESTful choreographies into Petri nets [80] that capture the REST' HATEOAS principle (see Section 2.6).

- *RESTful decision service.* Chapter 8 brings forth the peculiarities of implementing choreographies' exclusive gateways before proposing the Decision Model and Notation (DMN) standard [67] as a solution for solving the problem of data misinterpretation. The outcome is the introduction of a RESTful decision service that is integrated with RESTful choreographies to eliminate the misinterpretation of the decision. Therefore, the involved participants are driven to follow the right paths after exclusive gateways. In addition, using DMN alongside RESTful choreographies improves their comprehensibility and maintainability [5].

- *ChoreoGuide.* RESTful choreography guide, presented in Chapter 9 represents an hybrid service (a partial orchestration service) that guides the choreography participants towards a successful interaction. The ChoreoGuide includes a REST resource model (collectively agreed by the involved participants) that is needed for guiding the participants. The ChoreoGuide is not a full orchestration service because it does not call other services for reaching its own goal [79]. It is a reactive service that checks the validity of the requests (based on the REST resource state) and their order before forwarding it to the original recipient. In the presence of the ChoreoGuide, RESTful choreography participants do not have to switch their client and server roles to send notifications to each other. They are all clients with respect to the ChoreoGuide RESTful server. This also allows participants who do not have complex systems in place to be involved in the choreography. ChoreoGuides can be used as a starting point for designing third party Web services that specialize in applying a platform business model, like Airbnb[1], Uber[2], or Easychair[3].

- *REST Annotator tool and a comprehensive evaluation.* Last but not least, we developed a tool, called *REST Annotator*, that automatically derives RESTful choreographies from business process choreographies. The tool employees the advanced derivation approach described in Chapter 6. The derivation approach is evaluated, in Chapter 10, by using the *REST Annotator* tool on 172 choreography models, containing 864 choreography tasks that have syntactically-correct labels. The results show that 89.35% of the

1 https://www.airbnb.com/
2 https://www.uber.com/de/de/
3 http://www.easychair.org/

REST verbs are derived correctly, while the correctly derived REST request URIs reach 93.65% of total derivations.

11.2 LIMITATIONS AND FUTURE WORK

In this thesis we address the conceptual gap between process choreographies and RESTful interactions. We argue that this thesis' contributions significantly narrow down this gap, but, however, the gap is not considered closed. In this section we list some limitations with respect to some of our research contributions and propose future work to further close the conceptual gap.

One particular limitation, when it comes to implement concrete instances of a RESTful choreography model, is the instance correlation [12], especially the correlation between the REST resources. This is complicated as resource instances are reflected in their URIs. Future works needs to investigate solutions that relate choreography instances with resource instances and resource instances with each other.

Our derivation approach has also its limitations, which are grounded in the imprecise nature of natural language and the capabilities of the employed language processing tools. This imprecision is an important cause for several incorrectly identified REST verbs and REST URIs, which have to be corrected by REST experts. A particular limitation is related to the labeling style. If too many nouns are used it is hard to identify the intended action and business object. For example, the label *application letter submission* would yield *PUT letter/applied* instead of the more preferable *PUT applicationLetter/submitted*. Future work can address these limitations by making use of word sense disambiguation technology and the behavioral aspects of the choreography diagram. Word sense disambiguation utilizes external knowledge repositories such as WordNet [53] or BabelNet [55] together with contextual information or speech acts [9, 47] in order to identify the correct interpretation of a word. Its usefulness has already been investigated for process models in [81]. Behavioural aspects relate to the sequential order of choreography tasks [98]. The fact that only certain sequences and combinations of messages make sense can be used to describe constraints that restrict the number of potential interpretations [38]. For example, if a POST and a GET request have been identified and the respective choreography task is at the beginning of the interaction, then it is more likely to be a POST request. Furthermore, the derivation approach does not consider messages and their labeling. Including them may result in an increase of the URI generation accuracy as the messages may help to better identify the business object passed to the recipient.

Implementing RESTful choreographies by concrete services is not fully investigated. The RESTful decision service and ChoreoGuide are introduced as third party services that facilitates the enactment of RESTful choreographies. In an ideal setting, choreography participants have

to autonomously interact with each other without the need of a third party service. As future work we propose to consider the sixth and the only optional REST constraint—code-on-demand (see Section 2.6). This may allow the participants to share executable code between one other with the purpose of enforcing the choreography's control flow and message payloads.

BIBLIOGRAPHY

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer, 2004. (Cited on page 9.)

[2] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005. (Cited on page 29.)

[3] Alistair Barros, Gero Decker, Marlon Dumas, and Franz Weber. *Fundamental Approaches to Software Engineering: 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences, on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007. Proceedings*, chapter Correlation Patterns in Service-Oriented Architectures, pages 245–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-71289-3. doi: 10.1007/978-3-540-71289-3_20. URL http://dx.doi.org/10.1007/978-3-540-71289-3_20. (Cited on page 30.)

[4] Alistair Barros, Thomas Hettel, and Christian Flender. Process choreography modeling. In *Handbook on Business Process Management 1*, pages 257–277. Springer, 2010. (Cited on page 14.)

[5] Kimon Batoulis, Andreas Meyer, Ekaterina Bazhenova, Gero Decker, and Mathias Weske. Extracting decision logic from process models. In Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson, editors, *Advanced Information Systems Engineering*, pages 349–366, Cham, 2015. Springer International Publishing. ISBN 978-3-319-19069-3. (Cited on pages 105 and 134.)

[6] Betty Birner and Gregory Ward. Uniqueness, familiarity, and the definite article in english. In *Annual Meeting of the Berkeley Linguistics Society*, volume 20, pages 93–102, 1994. (Cited on page 73.)

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. W3c note, WWW Consortium, March 2001. URL http://www.w3.org/TR/wsdl. (Cited on page 28.)

[8] David Raymond Christiansen, Marco Carbone, and Thomas Hildebrandt. Formal semantics and implementation of bpmn 2.0 inclusive gateways. In Mario Bravetti and Tevfik Bultan, editors, *Web Services and Formal Methods*, pages 146–160, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19589-1. (Cited on page 53.)

[9] William W Cohen, Vitor R Carvalho, and Tom M Mitchell. Learning to classify email into" speech acts". In *EMNLP*, volume 4, pages 309–316, 2004. (Cited on page 135.)

[10] Mario Cortes-Cornax, Sophie Dupuy-Chessa, Dominique Rieu, and Marlon Dumas. Evaluating Choreographies in BPMN 2.0 Using an Extended Quality Framework. In Remco Dijkman, Jörg Hofstetter, and Jana Koehler, editors, *Business Process Model and Notation*, pages 103–117, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25160-3. (Cited on page 42.)

[11] Chris Dannen. *Introducing Ethereum and Solidity*. Springer, 2017. ISBN 978-1-484-22534-9. (Cited on page 28.)

[12] G. Decker and M. Weske. Instance isolation analysis for service-oriented architectures. In *2008 IEEE International Conference on Services Computing*, volume 1, pages 249–256, July 2008. doi: 10.1109/SCC.2008.44. (Cited on pages 30 and 135.)

[13] Gero Decker and Alistair Barros. Interaction modeling using bpmn. In Arthur ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, pages 208–219, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78238-4. (Cited on page 14.)

[14] Gero Decker and Mathias Weske. *Business Process Management: 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007. Proceedings*, chapter Local Enforceability in Interaction Petri Nets, pages 305–319. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-75183-0. doi: 10.1007/978-3-540-75183-0_22. URL http://dx.doi.org/10.1007/978-3-540-75183-0_22. (Cited on page 28.)

[15] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 296–303. IEEE, 2007. (Cited on page 28.)

[16] Gero Decker, Oliver Kopp, Frank Leymann, Kerstin Pfitzner, and Mathias Weske. Modeling service choreographies using bpmn and bpel4chor. In Zohra Bellahsène and Michel Léonard, editors, *Advanced Information Systems Engineering*, pages 79–93, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-69534-9. (Cited on page 28.)

[17] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Interacting services: From specification to execution. *Data & Knowledge Engineering*, 68(10):946 – 972, 2009. ISSN 0169-023X. doi: https://doi.org/10.1016/j.datak.2009.04.003. URL http://www.sciencedirect.com/science/article/pii/S0169023X09000457. (Cited on page 13.)

[18] Christian Denger, Daniel M. Berry, and Erik Kamsties. Higher quality requirements specifications through natural language patterns. In *2003 IEEE Int. Conference on Software - Science, Technology and Engineering*, pages 80–90, 2003. (Cited on page 67.)

[19] Marlon Dumas, Alexander Grosskopf, Thomas Hettel, and Moe Wynn. Semantics of standard process models with or-joins. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 41–58, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76848-7. (Cited on page 53.)

[20] Gregor Engels, Roland Hücking, Stefan Sauer, and Annika Wagner. UML Collaboration Diagrams and Their Transformation to Java. In Robert France and Bernhard Rumpe, editors, *«UML»'99 — The Unified Modeling Language*, pages 473–488, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-46852-3. (Cited on page 29.)

[21] Gregor Engels, Reiko Heckel, Jochen M. Küster, and Luuk Groenewegen. Consistency-preserving model evolution through transformations. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002 — The Unified Modeling Language*, pages 212–227, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45800-5. (Cited on page 79.)

[22] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999. (Cited on page 43.)

[23] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. (Cited on pages 9, 21, and 99.)

[24] Alain Finkel. The minimal coverability graph for petri nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993*, pages 210–243, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. ISBN 978-3-540-47631-3. (Cited on page 86.)

[25] Organization for the Advancement of Structured Information Standards (OASIS). Web services business process execution language (ws-bpel) 2.0. *OASIS Standards*, 04 2007. (Cited on pages 13, 14, and 28.)

[26] The Open Group. Service-Oriented Architecture Ontology, Version 2.0. https://publications.opengroup.org/standards/soa/c144, April 2014. (Cited on pages xiii, 3, 9, 10, and 110.)

[27] Stephan Haarmann, Kimon Batoulis, Adriatik Nikaj, and Mathias Weske. Dmn decision execution on the ethereum blockchain. In John Krogstie and Hajo A. Reijers, editors, *Advanced Information Systems Engineering*, pages 327–341, Cham, 2018. Springer International Publishing. ISBN 978-3-319-91563-0. (Cited on page 105.)

[28] Florian Haupt, Frank Leymann, and Cesare Pautasso. A conversation based approach for modeling REST APIs. In *Proc. of the 12th Working IEEE / IFIP Conference on Software Architecture (WICSA 2015)*, Montreal, Canada, May 2015. (Cited on pages 30 and 41.)

[29] Internet Engineering Task Force (IETF). Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. https://tools.ietf.org/html/rfc7231, June 2014. (Cited on pages 49 and 68.)

[30] Ana Ivanchikj, Cesare Pautasso, and Silvia Schreier. Visual modeling of restful conversations with restalk. *Journal of Software & Systems Modeling*, pages 1–21, May 2016. ISSN 1619-1366. doi: 10.1007/s10270-016-0532-2. URL http://link.springer.com/article/10.1007/s10270-016-0532-2. (Cited on pages 4 and 31.)

[31] Nickolaos Kavantzas. Web services choreography description language (ws-cdf) version 1.0. *http://www. w3. org/TR/ws-cdl-10/*, 2004. (Cited on page 28.)

[32] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. *W3C candidate recommendation*, 9:290–313, 2005. (Cited on page 13.)

[33] Andres Knöpfel, Bernhard Gröne, and Peter Tabeling. Fundamental modeling concepts. *Effective Communication of IT Systems, England*, 2005. (Cited on page 125.)

[34] Peter Kolb. Disco: A multilingual database of distributionally similar words. *Proceedings of KONVENS-2008, Berlin*, 2008. (Cited on pages 66 and 125.)

[35] Peter Kolb. Experiments on the difference between semantic similarity and relatedness. In *Proc. of the 17th Nordic Conference on Comp. Linguistics*, 2009. (Cited on page 66.)

[36] M. Laitkorpi, P. Selonen, and T. Systa. Towards a model-driven process for designing restful web services. In *2009 IEEE International Conference on Web Services*, pages 173–180, July 2009. doi: 10.1109/ICWS.2009.63. (Cited on page 30.)

[37] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. Generating natural language texts from business process models. In *Proc. of the 24th Int. Conference on Advanced Information Systems Engineering*, pages 64–79, 2012. (Cited on page 67.)

[38] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco Dijkman, and Heiner Stuckenschmidt. Probabilistic optimization of semantic process model matching. *Business Process Management*, pages 319–334, 2012. (Cited on page 135.)

[39] Henrik Leopold, Rami-Habib Eid-Sabbagh, Jan Mendling, Leonardo Guerreiro Azevedo, and Fernanda Araujo Baião. Detection of naming convention violations in process models for different languages. *Decision Support Systems*, 56:310–325, 2013. (Cited on pages 62, 72, and 125.)

[40] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. Supporting process model validation through natural language generation. *IEEE Trans. Software Eng.*, 40(8):818–840, 2014. (Cited on page 67.)

[41] Frank Leymann. *Web Services Flow Language (WSFL 1.0)*. BM Software Group, 05 2001. (Cited on page 14.)

[42] Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998. (Cited on page 66.)

[43] Niels Lohmann, Eric Verbeek, and Remco Dijkman. *Petri Net Transformations for Business Processes – A Survey*, pages 46–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00899-3. doi: 10.1007/978-3-642-00899-3_3. URL https://doi.org/10.1007/978-3-642-00899-3_3. (Cited on page 84.)

[44] Christopher Lyons. *Defining definiteness*, page 253–281. Cambridge Textbooks in Linguistics. Cambridge University Press, 1999. doi: 10.1017/CBO9780511605789.008. (Cited on pages 72, 73, and 74.)

[45] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. Formal semantics and analysis of bpmn process models using petri nets. 12 2018. (Cited on pages 20 and 84.)

[46] Mark Masse. *REST API design rulebook*. O'Reilly Media, Inc., 2011. (Cited on pages 42 and 100.)

[47] Raul Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. The action workflow approach to workflow management technology. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 281–288. ACM, 1992. (Cited on page 135.)

[48] Jan Mendling and Michael Hafner. From WS-CDL choreography to BPEL process orchestration. *Journal of Enterprise Information Management (JEIM)*, pages 506–515, 2008. (Cited on page 28.)

[49] Jan Mendling, Kristian Bisgaard Lassen, and Uwe Zdun. On the transformation of control flow between block-oriented and graph-oriented process modelling languages. *IJBPIM*, 3(2):96–108, 2008. doi: 10.1504/IJBPIM.2008.020973. URL https://doi.org/10.1504/IJBPIM.2008.020973. (Cited on page 28.)

[50] Jan Mendling, Hajo A. Reijers, and Jan Recker. Activity labeling in process modeling: Empirical insights and recommendations. *Inf. Syst.*, 35 (4):467–482, 2010. (Cited on page 61.)

[51] Jan Mendling, Ingo Weber, Wil Van Der Aalst, Jan Vom Brocke, Cristina Cabanillas, Florian Daniel, Søren Debois, Claudio Di Ciccio, Marlon Dumas, Schahram Dustdar, Avigdor Gal, Luciano García-Bañuelos, Guido Governatori, Richard Hull, Marcello La Rosa, Henrik Leopold, Frank Leymann, Jan Recker, Manfred Reichert, Hajo A. Reijers, Stefanie Rinderle-Ma, Andreas Solti, Michael Rosemann, Stefan Schulte, Munindar P. Singh, Tijs Slaats, Mark Staples, Barbara Weber, Matthias Weidlich, Mathias Weske, Xiwei Xu, and Liming Zhu. Blockchains for business process management - challenges and opportunities. *ACM Trans. Manage. Inf. Syst.*, 9(1):4:1–4:16, February 2018. ISSN 2158-656X. doi: 10.1145/3183367. URL http://doi.acm.org/10.1145/3183367. (Cited on page 29.)

[52] Andreas Meyer, Luise Pufahl, Kimon Batoulis, Dirk Fahland, and Mathias Weske. Automating Data Exchange in Process Choreographies. *Information Systems*, 2015. URL http://bpt.hpi.uni-potsdam.de/pub/Public/AndreasMeyer/Automating_Data_Exchange_in_Process_Choreographies_J.pdf. in press. (Cited on pages 29 and 30.)

[53] G. A. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995. (Cited on pages 65, 125, and 135.)

[54] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008. (Cited on pages 4 and 28.)

[55] Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012. (Cited on page 135.)

[56] Adriatik Nikaj and Mathias Weske. Formal specification of restful choreography properties. In Alessandro Bozzon, Philippe Cudre-Maroux, and Cesare Pautasso, editors, *Web Engineering*, pages 365–372, Cham,

2016. Springer International Publishing. ISBN 978-3-319-38791-8. (Cited on pages 80 and 96.)

[57] Adriatik Nikaj, Sankalita Mandal, Cesare Pautasso, and Mathias Weske. From choreography diagrams to restful interactions. In *Engineering Service Oriented Applications WESOA'15, co-located with ICSOC 2015*, Springer, 2015. Springer. (Cited on pages 39 and 67.)

[58] Adriatik Nikaj, Fabian Pittke, Mathias Weske, and Jan Mendling. Semi-automatic derivation of restful interactions from choreography diagrams. In Rainer Schmidt, Wided Guédria, Ilia Bider, and Sérgio Guerreiro, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 141–156, Cham, 2016. Springer International Publishing. ISBN 978-3-319-39429-9. (Cited on pages 58 and 70.)

[59] Adriatik Nikaj, Marcin Hewelt, and Mathias Weske. Towards implementing rest-enabled business process choreographies. In Witold Abramowicz and Adrian Paschke, editors, *Business Information Systems*, pages 223–235, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93931-5. (Cited on page 108.)

[60] Adriatik Nikaj, Mathias Weske, and Jan Mendling. Semi-automatic derivation of restful choreographies from business process choreographies. *Software & Systems Modeling*, 18(2):1195–1208, Apr 2019. ISSN 1619-1374. (Cited on pages 58 and 125.)

[61] Jörg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann. Bpellight. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, pages 214–229, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75183-0. (Cited on page 14.)

[62] OMG. Unified Modeling Language (UML), Version 1.1. https://www.omg.org/spec/UML/1.1/About-UML/, December 1997. (Cited on page 29.)

[63] OMG. Unified Modeling Language (UML), Version 2.0. http://www.omg.org/spec/UML/2.0/, July 2005. (Cited on pages 29, 30, and 87.)

[64] OMG. Object Constraint Language (OCL), Version 2.0. http://www.omg.org/spec/OCL/2.0/, May 2006. (Cited on pages 112 and 114.)

[65] OMG. Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0/, January 2011. (Cited on pages 3, 10, 18, 27, 41, 46, 58, 61, 97, and 112.)

[66] OMG. XML Metadata Interchange (XMI) Specification, Version 2.5.1. https://www.omg.org/spec/XMI/2.5.1/, June 2015. (Cited on page 48.)

[67] OMG. Decision Model and Notation, Version 1.0, September 2015. (Cited on pages 96, 98, and 134.)

[68] OMG. Meta Object Facility (MOF), Version 2.5.1. https://www.omg.org/spec/MOF/2.5.1/, November 2016. (Cited on page 35.)

[69] Steve Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. *AI Matters*, 1(2):19–21, December 2014. ISSN 2372-3483. doi: 10.1145/2685328.2685334. URL http://doi.acm.org/10.1145/2685328.2685334. (Cited on page 28.)

[70] Alexander Osterwalder et al. The business model ontology: A proposition in a design science approach. 2004. (Cited on page 7.)

[71] Chun Ouyang, Marlon Dumas, HM Arthur, and Wil Mp Van Der Aalst. Pattern-based translation of bpmn process models to bpel web services. *International Journal of Web Services Research (IJWSR)*, 5(1):42–62, 2008. (Cited on page 28.)

[72] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. From business process models to process-oriented software systems. *ACM Trans. Softw. Eng. Methodol.*, 19(1):2:1–2:37, August 2009. ISSN 1049-331X. doi: 10.1145/1555392.1555395. URL http://doi.acm.org/10.1145/1555392.1555395. (Cited on page 28.)

[73] Francis Palma, Javier Gonzalez-Huerta, Naouel Moha, Yann-Gaël Guéhéneuc, and Guy Tremblay. *Service-Oriented Computing: 13th International Conference, ICSOC 2015, Goa, India, November 16-19, 2015, Proceedings*, chapter Are RESTful APIs Well-Designed? Detection of their Linguistic (Anti)Patterns, pages 171–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48616-0. doi: 10.1007/978-3-662-48616-0_11. URL http://dx.doi.org/10.1007/978-3-662-48616-0_11. (Cited on page 100.)

[74] Francis Palma, Javier Gonzalez-Huerta, Naouel Moha, Yann-Gaël Guéhéneuc, and Guy Tremblay. Are restful apis well-designed? detection of their linguistic (anti)patterns. In *Service-Oriented Computing*, Lecture Notes in Computer Science. Springer, 2015. ISBN 978-3-662-48615-3. doi: 10.1007/978-3-662-48616-0_11. URL http://dx.doi.org/10.1007/978-3-662-48616-0_11. (Cited on page 132.)

[75] Cesare Pautasso. Rest vs. ws-* comparison. 2009. (Cited on pages 3 and 9.)

[76] Cesare Pautasso. BPMN for REST. In *Proc. of the 3rd International Business Process Modeling Notation Workshop (BPMN 2011)*, pages 74–87, Lucerne, Switzerland, November 2011. doi: 10.1007/978-3-642-25160-3_6. (Cited on pages 4, 31, and 41.)

[77] Cesare Pautasso and Erik Wilde. Push-enabling restful business processes. In *International Conference on Service-Oriented Computing*, pages 32–46. Springer, 2011. (Cited on pages 4, 25, 31, 41, 44, 83, 120, and 130.)

[78] Cesare Pautasso, Ana Ivanchikj, and Silvia Schreier. Modeling RESTful Conversations with extended BPMN Choreography diagrams. In *ECSA 2015*. Springer. (Cited on pages 4 and 31.)

[79] Chris Peltz. Web services orchestration and choreography. *Computer*, (10):46–52, 2003. (Cited on pages 108 and 134.)

[80] Carl Adam Petri. Kommunikation mit automaten. 1962. (Cited on pages 19, 84, and 134.)

[81] Fabian Pittke, Henrik Leopold, and Jan Mendling. Automatic detection and resolution of lexical ambiguity in process models. *IEEE Trans. Software Eng.*, 41(6):526–544, 2015. doi: 10.1109/TSE.2015.2396895. URL https://doi.org/10.1109/TSE.2015.2396895. (Cited on page 135.)

[82] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997. (Cited on page 67.)

[83] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proc. of the 14th Int. Joint Conference on Artificial Intelligence*, pages 448–453, 1995. (Cited on page 66.)

[84] Paul Rimba, An Binh Tran, Ingo Weber, Mark Staples, Alexander Ponomarev, and Xiwei Xu. Comparing blockchain and cloud services for business process execution. In *2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017*, pages 257–260, 2017. doi: 10.1109/ICSA.2017.44. URL https://doi.org/10.1109/ICSA.2017.44. (Cited on page 29.)

[85] Silvia Schreier. Modeling restful applications. In *Proceedings of the second international workshop on restful design*, pages 15–21. ACM, 2011. (Cited on page 30.)

[86] Christoph Schroeder. Articles and article systems in some areas of europe. *EMPIRICAL APPROACHES TO LANGUAGE TYPOLOGY*, 20(8): 545, 2006. (Cited on page 73.)

[87] Nigel Slack, Stuart Chambers, and Robert Johnston. *Operations management*. Pearson education, 2010. (Cited on page 9.)

[88] Paul P. Tallon. A process-oriented perspective on the alignment of information technology and business strategy. *Journal of Management Information Systems*, 24(3):227–268, 2007. doi: 10.2753/MIS0742-1222240308. URL https://doi.org/10.2753/MIS0742-1222240308. (Cited on page 3.)

[89] Satish Thatte. Xlang: Web services for business process design. *Microsoft Corporation*, 2001, 2001. (Cited on page 14.)

[90] Francisco Valverde and Oscar Pastor. Dealing with rest services in model-driven web engineering methods. *V Jornadas Científico-Técnicas en Servicios Web y SOA, JSWEB*, 2009. (Cited on page 30.)

[91] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8:21–66, 1998. (Cited on page 86.)

[92] Wil M. P. van der Aalst. *Process Discovery: An Introduction*, pages 125–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-19345-3. doi: 10.1007/978-3-642-19345-3_5. URL https://doi.org/10.1007/978-3-642-19345-3_5. (Cited on page 9.)

[93] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business process management: A survey. In Wil M. P. van der Aalst and Mathias Weske, editors, *Business Process Management*, pages 1–12, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-44895-2. (Cited on page 7.)

[94] Wil MP Van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997. (Cited on page 20.)

[95] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998. (Cited on page 20.)

[96] Hagen Völzer. A new semantics for the inclusive converging gateway in safe processes. In Richard Hull, Jan Mendling, and Stefan Tai, editors, *Business Process Management*, pages 294–309, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15618-2. (Cited on page 53.)

[97] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. *Untrusted Business Process Monitoring and Execution Using Blockchain*, pages 329–347. Springer International Publishing, Cham, 2016. ISBN 978-3-319-45348-4. doi: 10.1007/978-3-319-45348-4_19. URL http://dx.doi.org/10.1007/978-3-319-45348-4_19. (Cited on page 28.)

[98] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410–429, 2011. doi: 10.1109/TSE.2010.96. URL https://doi.org/10.1109/TSE.2010.96. (Cited on page 135.)

[99] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012. ISBN 978-3-642-28615-5. (Cited on pages xiii, 3, 7, 8, 37, 42, and 80.)

[100] W3C. *Simple Object Access Protocol (SOAP) 1.2*. World Wide Web Consortium, 2003. (Cited on pages 3, 9, and 28.)

[101] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proc. of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, 1994. (Cited on page 66.)

[102] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen. The blockchain as a software connector. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 182–191, April 2016. doi: 10.1109/WICSA.2016.21. (Cited on page 29.)

[103] Xiwei Xu, Liming Zhu, Udo Kannengiesser, and Yan Liu. An architectural style for process-intensive web information systems. In Lei Chen, Peter Triantafillou, and Torsten Suel, editors, *Web Information Systems Engineering – WISE 2010*, pages 534–547, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-17616-6. (Cited on pages 4 and 31.)

[104] Johannes Maria Zaha, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Let's dance: A language for service behavior modeling. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 145–162, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-48289-5. (Cited on pages 14 and 27.)

[105] Johannes Maria Zaha, Marlon Dumas, Arthur Ter Hofstede, Alistair Barros, and Gero Decker. Service interaction modeling: Bridging global and local views. In *Enterprise Distributed Object Computing Conference, 2006. EDOC'06. 10th IEEE International*, pages 45–55. IEEE, 2006. (Cited on pages 16 and 27.)

[106] J. Ziemann and J. Mendling. EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In *International Conference "Modern Information Technology in the Innovation Processes of the Industrial Enterprises*, Genova, Italy, 2005. (Cited on page 28.)

[107] Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson. Developing web services choreography standards—the case of rest vs. soap. *Decision Support Systems*, 40(1):9 – 29, 2005. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2004.04.008. URL http://www.sciencedirect.com/science/article/pii/S0167923604000612. Web services and process management. (Cited on page 3.)

All links were last followed on June 12, 2019.

DECLARATION

I hereby confirm that I have authored this thesis independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such. I am aware of the examination regulations and this thesis has not been previously submitted elsewhere.

Potsdam, June, 2019

Adriatik Nikaj