From Choreography Diagrams to RESTful Interactions

Adriatik Nikaj[1(✉)], Sankalita Mandal[1], Cesare Pautasso[2], and Mathias Weske[1]

[1] Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany
{adriatik.nikaj,sankalita.mandal,mathias.weske}@hpi.de
[2] Faculty of Informatics, University of Lugano (USI), Lugano, Switzerland
cesare.pautasso@usi.ch

Abstract. Today, business process management is a key approach to organize work, and many companies represent their operations in business process models. Recently, choreography diagrams have been introduced to represent interactions between business processes, run by different partners. 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. In this paper, a novel approach to enhance choreography diagrams by execution information is introduced. The approach is based on the REST architecture style, which is the primary way for interacting systems. Using enhanced choreography diagrams allows us to develop REST-based interactions among business partners in an efficient manner. The approach is illustrated by an example of an accommodation reservation service.

1 Introduction

As the number and the complexity of interactions between multiple business partners grow, it becomes important to describe them using precise models which can be refined to a level of detail suitable for their implementation. As more and more services adopt the representational state transfer (REST) architectural style [1] to expose their internal business processes on the Web [2], we identify the opportunity to define and represent interactions among business processes at a suitable level of abstraction with choreography diagrams, specified in the Business Process Model and Notation (BPMN) standard [3].

In this paper we introduce a novel approach to enrich BPMN choreography diagrams with annotations that facilitate capturing information required for an efficient implementation of REST-based information exchanges between two or more interacting business partners. The approach is motivated by an example of a multi-party conversation [4] inspired by Airbnb, a popular accommodation reservation service, which acts as a broker between many partners that are concurrently enacting business processes for offering, selecting and booking suitable accommodations.

The paper is organized as follows. The basics of BPMN choreography diagrams are introduced in Sect. 2, where also a motivating example is discussed.

A. Norta et al. (Eds.): ICSOC 2015 Workshops, LNCS 9586, pp. 3–14, 2016.
DOI: 10.1007/978-3-662-50539-7_1

Section 3 introduces the role of the REST architectural style during business process enactment. RESTful choreography diagrams are in the center of Sect. 4, providing the conceptual basis of the approach introduced. The approach is evaluated in Sect. 5 by applying the identified patterns to the multi-party interaction of the example. Section 6 explores related approaches; concluding remarks complete this paper.

2 Motivating Example

Choreography diagrams were first introduced in the standard BPMN 2.0 [3]. Going by the definition of a choreography, it does not focus on the internal activities of the participants, instead, it focuses on the interaction between the participants. Starting from a process model where each organization participating in the process is represented by separate pools, we can say that the choreography diagram corresponding to the process model will abstract all the work done by the individual pools and only portray the message exchanges between the pools.

The main building block of choreography diagram is a choreography task. A choreography task consists of three bands representing the two participants and the choreography task name. The participant who initiates the message exchange is named as initiator and is highlighted white whereas the other participant who receives the message and optionally sends a response is called the receiver and is highlighted in grey.

The messages sent by the initiator and/or the receiver can be shown explicitly using a message icon associated with the respective sender. The message icon must be unfilled if it is the initiating message of the choreography task. On the other hand, a message icon depicting the response must be highlighted in grey. Several choreography tasks can be modelled using a sub-choreography with more than two participant bands and a 'plus' sign indicating the abstraction.

An example is used both to illustrate choreography diagrams and to motivate our approach. Figure 1 presents a choreography diagram for the accommodation booking process of a fictional company ARS (Accommodation Reservation Service). There are four participants: 1. ARS (the main platform of communication), 2. Guest (who wants to rent an apartment), 3. Host (the owner of an apartment) and 4. Payment Organization (who handles the payment on behalf of ARS).

The conversation starts when the Guest makes a reservation request, represented by the first choreography task in Fig. 1. ARS checks whether the selected offer is still available at this point or not. The latter may happen if, while creating the reservation request, the apartment gets booked by another Guest, or the Host changes his plans. If the offer is not available any more, then ARS sends the Guest a message saying 'Not Possible' and the reservation request fails.

Otherwise, ARS notifies the Host about the request and waits for 24 h for a response. At this point, the Guest also receives an email that the Host has been notified and the Guest can expect an answer within 24 h. The Host can either accept or decline the request within 24 h and the Guest can cancel the request

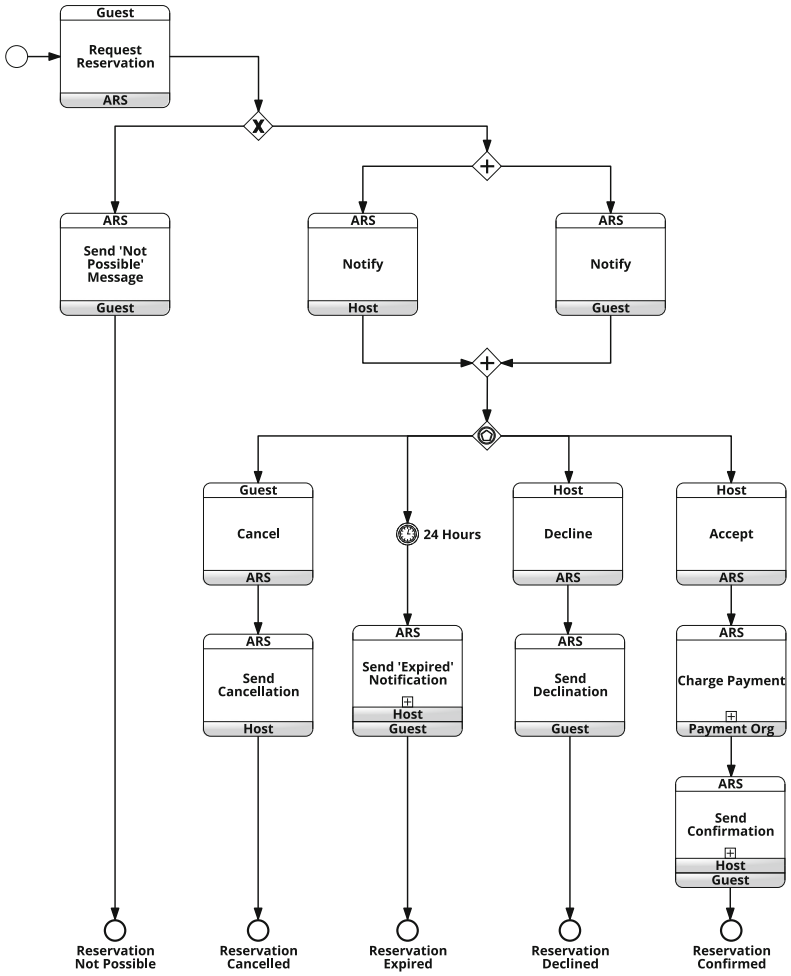

Fig. 1. Choreography diagram of an accommodation reservation service

before he gets a response from ARS. If none of these interactions happen within 24 h, the request automatically expires, and an expiration notification is sent to both Guest and Host.

In case of a cancellation by the Guest or a declination by the Host, the Host or the Guest is notified, respectively, and the choreography comes to an end. But when the Host accepts the request, the Payment Org comes into play. ARS requests the Payment Org to charge the Guest using the payment details entered at the time of creating the reservation request. After payment is completed, the reservation request is confirmed, and ARS sends this confirmation to both Guest and Host. In the example, the interactions between ARS and Payment Org are abstracted using the sub-choreography 'Charge Payment'.

As can be seen in the motivating example, a choreography diagram can have different gateways like exclusive gateways, event based gateways and parallel gateways. These gateways are introduced next.

The exclusive gateway is used in a choreography where based on a decision one of several alternatives can be chosen. Selecting one path deactivates the other options. In the example, when a Guest sends a reservation request to ARS, ARS checks the status of the Host's calendar and then one of two paths is chosen, i.e., either the Guest receives a 'Not Possible' message or the partners are notified about the reservation request.

An event based gateway is a branching point in a choreography where the alternatives are not chosen based on the data, rather, one of the alternatives is followed based on an event occurrence at that point. For example, after ARS sends the notification about the reservation request to the Host, it waits for any of the four events to occur: either the Host accepts the request or declines it within 24 h, the request expires after 24 h, or the Guest cancels the request. The event that occurs first determines the path to be followed.

Parallel gateways are used to represent independent paths that can be executed concurrently. In our example, if the reservation request is possible, then ARS sends notification to both the Host and the Guest concurrently.

3 RESTful Business Processes

The REST architectural style has gained widespread acceptance as the foundation for building Web-based applications and so-called RESTful Web services or RESTful Web APIs. These make full usage of the HTTP protocol not only to exchange messages between the participants (usually one client interacting with one or more Web service following a business process) but also to express properties (e.g., the addressing and identification of the Web resources involved in the interaction) and constraints (e.g., the idempotent request to perform a state transition) of the interactions. The result is a highly decentralized system, whose components are designed to be scalable under the assumption of stateless interactions and mostly read-only operations.

Web resources are uniquely and globally identified through URLs accessed via their Uniform Interface. In case of the HTTP protocol, this corresponds to a limited set of verbs (like GET, POST, PUT, DELETE) that define the effect of the basic interactions on the state of the resource. For example, POST will create a new resource whose identifier is determined by the server, while PUT will update (or create) a resource whose state and identifier are provided by the client. GET will retrieve the current state of the resource, while DELETE will remove it.

Business process models, seen as a collection of individual activities, can be used both to represent the orchestration of one or more Web resource as well as to indicate how the internal state of a resource may evolve. For example, a process may define the behavior of a client which looks for a suitable accommodation by navigating (GET) across multiple offerings, once an accommodation

has been found a client submits a reservation request (POST) and waits until either the request has been accepted by the host (PUT) or keeps looking for another accommodation.

Due to their client/server constraint, REST and HTTP do not play well with server-sent notifications of state changes, which typically rely on a publish/subscribe messaging system. Whereas it is possible to reverse the roles of the interaction and have the server call back the client (assuming the client also publishes a RESTful API for incoming notifications) or have the client poll the server by means of a feed subscription.

Concerning typical Web platforms such as ARS, this problem is solved by email-based notifications, whereby authenticated clients are associated with an email address, which is used to send event notifications triggered by state changes of the reservation system, or, more precisely, on the reservation resource. What is important is that the email does not only include information about the new state of the reservation resource, but also embeds hyperlinks so that the recipient may follow them to perform further interactions.

In general, when multiple clients are involved in an interaction with a resource shared between them, it becomes important to reason about their interactions not only from the perspective of individual participants, but also from the global perspective. Hence, we aim at creating RESTful choreography diagrams to reason about all possible valid interactions that lead to a given successful or unsuccessful outcome. These will help us model what interaction each participant may perform depending on the state of the shared resource, which may of course change and evolve as a consequence of the participants interactions.

4 Implementation of Choreographies

RESTful choreography diagrams are an enhancement of BPMN choreography diagram with REST-specific annotations. This section introduces the building elements of RESTful choreography diagrams and their uses, before causality relationship patterns are introduced.

4.1 REST Annotations

Since choreography tasks are the basic units for modeling interactions in a choreography, we use them for modeling the basic RESTful HTTP request/response interaction and email messaging. To properly annotate the choreography task with REST information, we use annotated messages associated to choreography tasks. This is realized by specializing the exchanged messages into HTTP request, HTTP response or email messages (see Fig. 2).

Using annotated messages for expressing REST specific information avoids the introduction of new modeling elements, and hence, avoids changing the metamodel of BPMN choreography diagrams.

Due to the fact that one of the architectural constraints of REST is the client/server communication, we identified two possible usages of the choreography task, which depend on the presence of a recipient's RESTful API.

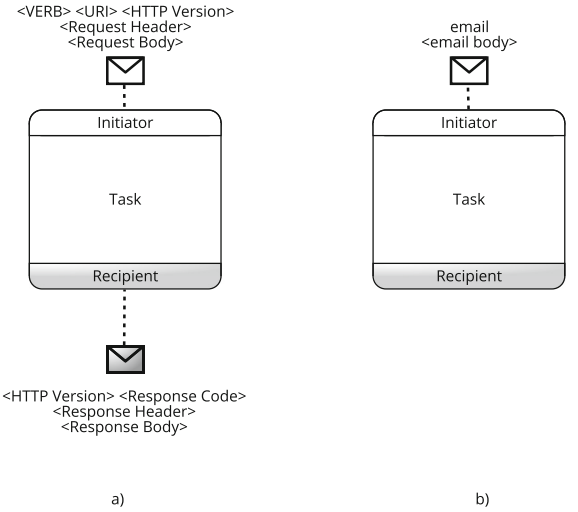

Fig. 2. Enriching choreography tasks by annotations

If the recipient provides a RESTful API then the client plays the role of the initiator, while the server plays the role of the recipient. The initiating message is a HTTP request and the return message is a HTTP response, as shown in Fig. 2(a).

As discussed earlier, the recipient may not always provide a RESTful API (e.g., the Guest and the Host in the example). The problem of notifying a client can be solved via an email-based approach. In this case, the server plays the role of the initiator, whereas the client plays the role of the recipient.

As initiating message, we use an email message, illustrated in Fig. 2(b), which contains further links for the client to follow in the upcoming choreography tasks. In this case, there is no need to model a return message.

The information in the HTTP requests/responses or email messages does not need to be of the same level of detail as that of the implementation, which might include, for instance, text fragments.

However, it is essential to include links which eventually determine the behavior of the entire interaction modeled in a choreography diagram. In the ARS scenario, for example, it is essential to include links in an email message to the Host that allows him to accept or decline the reservation request by following those links.

Sequence flow, per se, does not need any additional enhancement. However, we can make use of conditions, which can be added to conditional sequence flows. In the RESTful choreography diagram, the link can constitute a condition. It perfectly fulfills the requirement of the choreography diagram, which states that the condition should have been previously sent via a message (as links are). Conditional flows can be placed directly after choreography tasks as well as exclusive or inclusive gateways.

4.2 Causality Relationship Patterns

We observed several patterns that reoccur in choreography diagrams concerning the relation between the content of the messages exchanged and the ordering of choreography tasks. In REST-based interactions, links sent between participants pave the way for upcoming interactions. More specifically, the number of links contained in a choreography task impacts the type of the consecutive nodes in a choreography diagram.

In REST-based interaction scenarios, links are the most prominent artifacts, because links allow interaction partners to access and modify shared resources. The patterns introduced below categorize interactions based on the number of links transmitted. These categories provide useful information for checking if a RESTful choreography diagram is feasible.

- **no-link pattern** In case of the no-link pattern, the choreography task incorporates a HTTP response message or an email message without any link. 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 another choreography task, which does not have the same participants as the former interaction. In any case, the missing link hints to a lack of future conversation between the participants.
- **single-link pattern** In the single-link pattern, the choreography task incorporates a HTTP response message or an email message with a single link. This kind of message, generally, is a notification, which can link to additional information than what is included in the HTTP response body or the email body itself. Typical examples are a HTTP response linking to an updated resource and an email linking to the status of a recently changed resource.
- **multi-link pattern** In case of the multi-link pattern, the choreography task incorporates a HTTP response message or an email message with n ($n > 1$) links. This task can be followed by an exclusive gateway or an event-based gateway. In case of an exclusive gateway, the outgoing sequence flows of the gateway are conditional sequence flows, each of which refers to a link contained in the preceding choreography task. The following choreography task involve the client as initiator, making a request to the resource identified by the link corresponding to the chosen branch.

The validity of the patterns is illustrated as follows. If there is no link transmitted from the participant that provides a RESTful API, then applying the no-link pattern ensures that the other participants make no further interaction with the resource. A typical example of the no-link pattern in an online shopping scenario is a message that informs the customer of a late delivery.

If there is a single link transmitted among interaction partners, the receiver has a handle for further interactions. In an online shopping example, the customer receives an email message with a link to his open orders. Using this link, the customer can use GET messages to retrieve the current status of his orders.

If there is a gateway in a choreography diagram, the outcome of which leads to different resource states, then applying the multi-link pattern will make sure

that the links are transmitted beforehand. The customer might decide to cancel the order or update the delivery address, using several links that have been transmitted to him by the online shop.

5 Evaluation

To evaluate the effectiveness of the approach, we enhance the ARS choreography model with REST annotations and proceed one step further to the implementation level.

5.1 RESTful Choreography Diagram

In Sect. 4, patterns of RESTful interactions have been identified. We return to the motivating example described in Sect. 2 and apply the approach to the example; the resulting RESTful choreography diagram is shown in Fig. 3.

The approach is based on identifying the interaction types of choreography tasks, mentioned in Sect. 4. The first interaction type consists of an initiating message (HTTP request) and a return message (HTTP response), represented by the first choreography task in the diagram. The Guest makes a POST request and gets back the status '201 Created' along with a link (*/reservation/id/details*) for further correspondence.

The second type of interaction is an email-based notification sent from server side to the client. We can see this type of interaction when ARS notifies the Host about the reservation request. The Host receives an email with several links from ARS and sends no specific response at this point.

Regarding the causality relationship patterns introduced above, three scenarios can be encountered. If there is no link transmitted in the choreography task, then no further interaction between the participant and the resource is expected. This pattern can be seen in the example where the task 'Send Cancellation', which is implemented by an email based interaction without passing any link, leads to an end event (Reservation Cancelled).

The second pattern describes the choreography task containing a single link, which the participant can use to get further details about the resource. This pattern can be witnessed in several places. For example, when ARS notifies the Host about the reservation request, at the same time the Guest also receives an email with a single link */reservation/id/notified* allowing the Guest to get more details about the reservation at any time.

In general, whenever there is an update to a resource, using the single-link pattern, a link with the details is transmitted from ARS to other participants. This transmission is done either as an HTTP response (e.g., 'Accept') or as an email message (e.g., 'Send Confirmation', 'Send Declination').

The third pattern describes a scenario where a choreography task incorporating more than one link is followed by a gateway. This pattern is found when ARS sends a notification email to the Host containing three links. The task is followed by an event-based gateway. The Host can use the */reservation/id/details* link for

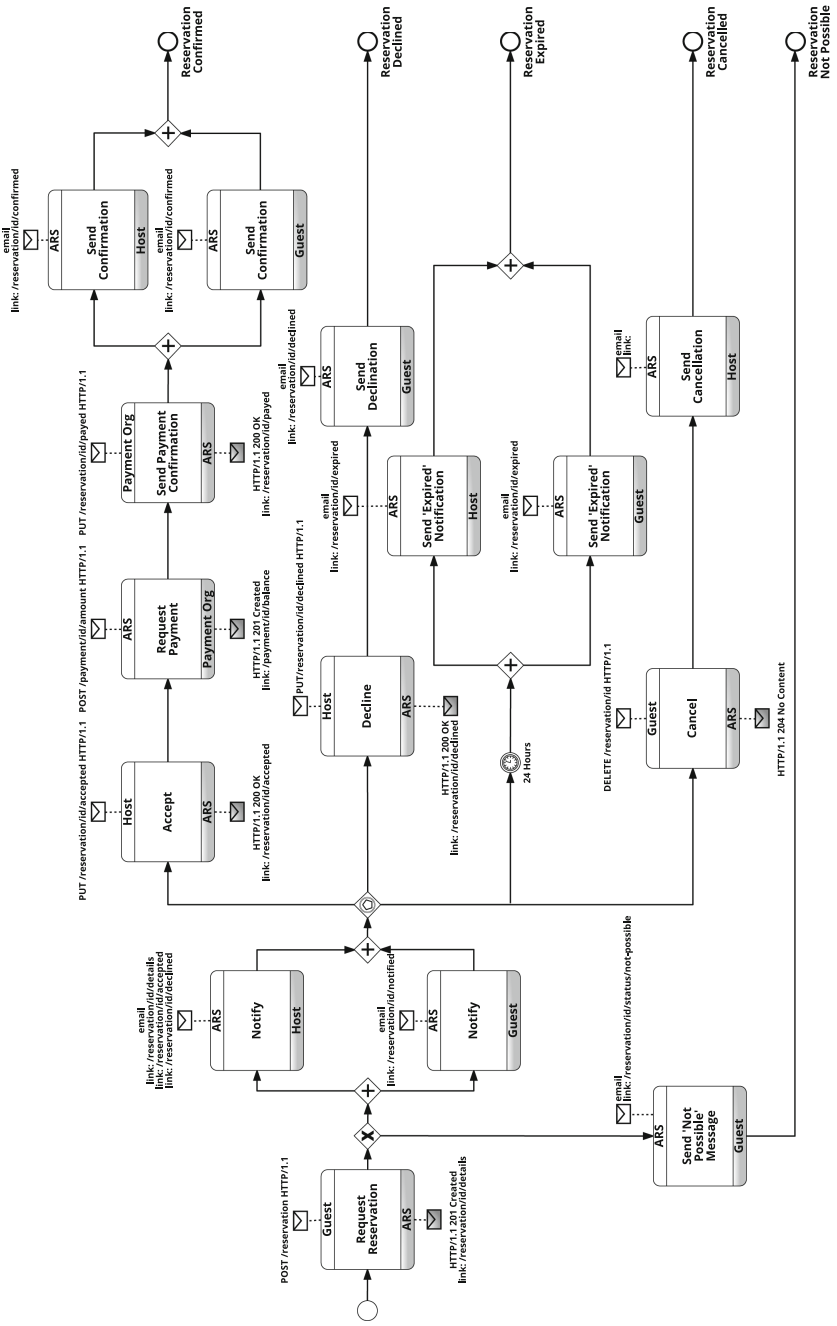

Fig. 3. RESTful choreography diagram for the motivating example

getting the detailed information about the reservation request. The other two links give the Host the options to accept or decline. This is represented in Fig. 3 by the tasks 'Accept' and 'Decline'. At this point, the choreography has two other possibilities: the reservation request can expire after 24 h or the Guest cancels the request. These interactions can be represented by an event-based gateway. In general, the multi-link pattern is used whenever there is an upcoming decision, represented by a branching structure in the choreography diagram.

5.2 From Model Level Towards Implementation Level

The goal of the approach is to mitigate the gap between model level and implementation level. So far, we have enriched models with annotations that provide information for the implementation level. In this section, we come up with a series of messages that can be generated from the RESTful choreography diagram.

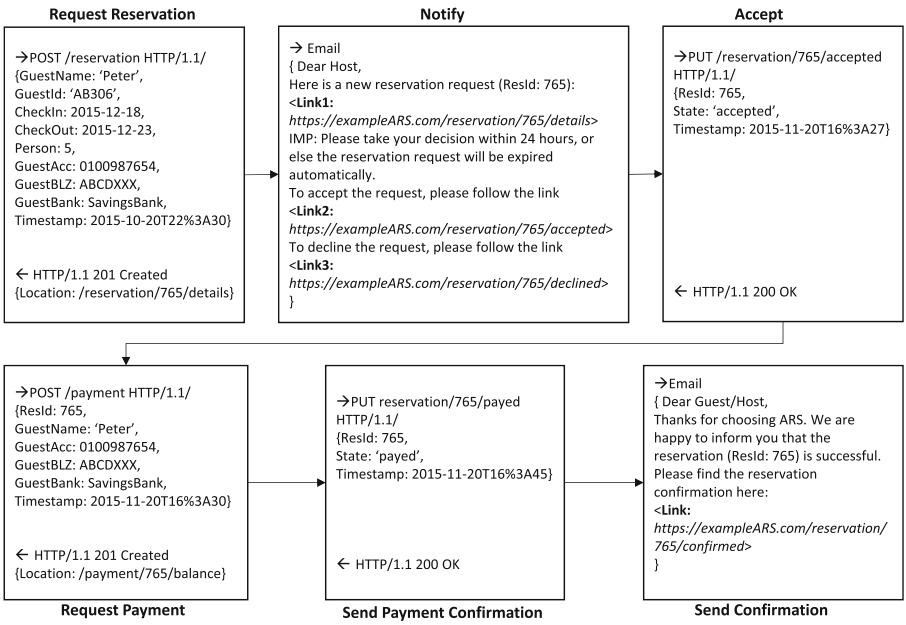

Fig. 4. Messages and their ordering resulting from an interaction between multiple participants, from the sending of a reservation request by a Guest to the message implementing the respective confirmation by ARS.

Figure 4 shows the messages generated from the interactions between ARS and the other three participants for one instance where the reservation is successful. The series of messages identifies with the REST and email interactions starting from the creation of reservation request by the Guest until the reservation is confirmed. Each interaction is represented by a single rectangle containing

the generic message template for each REST operation and the corresponding responses. The rectangles are named with the same labels used as choreography tasks in the RESTful choreography model.

The reservation request contains details about the inquired reservation as well as the payment details of the Guest. The reservation details are then stored in the location */reservation/765/details* where 765 is the unique reservation id. The link https://exampleARS.com/reservation/765/details[1] is passed to the Host along with two other links for accepting or declining the request, all included in an email sent by ARS, taking advantage of the multi-link pattern.

In this case, the Host decides to accept the Guest by clicking on the link https://exampleARS.com/reservation/765/accepted. After getting the acceptance from the Host, ARS creates a payment request using the POST operation with the parameters containing the same payment details entered by the Guest at the time of creating reservation request. Finally after a successful payment, an email is sent to both the Guest and the Host with the link https://exampleARS.com/reservation/765/confirmed, using the single link pattern.

In standard BPMN or choreography models, this information cannot be represented. With our approach, it is possible to define this information and to integrate it in the model. Moreover, including implementation information in the model not only helps to come up with the messages for each instance, but also gives a detailed overview of all possible instances.

6 Related Work

Different approaches exist for bridging the gap between the choreographies and their implementation. One notable approach is that of Decker et al. [5], in which the authors extend the BPEL web service composition standard [6] for modeling choreographies. The result is BPEL4Chor, an extended language capable of orchestrating choreographies. That paper uses a bottom-up approach that integrates pre-exisiting BPEL service orchestrations, based on web services standards like SOAP and WSDL [7]. In contrast, the approach presented in this paper uses a top-down approach, starting at the choreography level. In addition, our technological basis is REST as opposed to standard web services techniques, and we incorporate email messaging, which is not addressed in the BPEL4Chor approach.

Another approach, which enhances a modeling language for bridging the gap between modeling and implementation is BPMN for REST [8]. The author introduces an enrichment of BPMN for modeling RESTful business processes. The paper's focus is on the representation of business processes that interact with external resources or where elements of the business processes are published as resources. Conversely, our approach abstracts from business processes or their composing elements and focuses only on the interaction aspect of business processes. Nevertheless, these two approaches are complementary, and thus, can be both used for modeling the same scenario from the two respective perspectives. The mapping between the two representations is left as a future work.

[1] exampleARS.com is a fictional website.

7 Conclusions

This paper introduced a novel approach for enriching BPMN choreography diagrams with REST-specific annotations, aiming at easing the transition from choreography models to the implementation of RESTful HTTP conversations. RESTful choreography diagrams still comply with the formal specification of BPMN standard, since only annotations are used to capture the additional information. This approach makes it feasible to use existing modeling tools and the standardized output serialization, which is also defined in the BPMN standard.

This paper provides the conceptual basis of the approach and uses an example that is inspired by a well-known accommodation reservation service. The patterns identified can also be found in other scenarios, including online shopping and e-commerce scenarios [9], where the integration of HTTP request-response interactions and email communication plays an important role.

Future work will include an investigation on methodological aspect of the approach, in particular which stakeholders are involved, to further ease the transition from choreography models to executable REST based implementation.

References

1. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis AAI9980887(2000)
2. Pautasso, C., Wilde, E.: Push-enabling RESTful business processes. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) Service Oriented Computing. LNCS, vol. 7084, pp. 32–46. Springer, Heidelberg (2011)
3. OMG: Business Process Model and Notation (BPMN), Version 2.0, January 2011. http://www.omg.org/spec/BPMN/2.0/
4. Hohpe, G.: Let's have a conversation. IEEE Internet Comput. **11**(3), 78–81 (2007)
5. Decker, G., Kopp, O., Leymann, F., Weske, M.: Bpel4chor: extending bpel for modeling choreographies. In: IEEE International Conference on Web Services, ICWS 2007, pp. 296–303. IEEE (2007)
6. Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., et al.: Web services business process execution language version 2.0. OASIS standard 11, 10 (2007)
7. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer, New York (2004)
8. Pautasso, C.: BPMN for REST. In: Dijkman, R., Hofstetter, J., Koehler, J. (eds.) BPMN 2011. LNBIP, vol. 95, pp. 74–87. Springer, Heidelberg (2011)
9. Benatallah, B., Casati, F., et al.: Web service conversation modeling: A cornerstone for e-business automation. IEEE Internet Comput. **8**(1), 46–54 (2004)