# Formal Specification of RESTful Choreography Properties

Adriatik Nikaj(✉) and Mathias Weske

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{adriatik.nikaj,mathias.weske}@hpi.de

**Abstract.** BPM community has developed a rich set of languages for modeling interactions. In previous work, we argue that business process choreographies are suited for modeling REST-based interactions. To this end, RESTful choreographies have been introduced as an extension of business process choreographies. However, RESTful choreographies do not provide information about the validity of interactions. In this paper, we introduce formal completeness properties. These properties support developers to verify REST-based interactions. The approach is motivated by an example of an examination procedure in the context of a massive open online course.

## 1 Introduction

With the surging use of REST architectural style [1], there is a need to model REST-based interactions from a global perspective. RESTful choreography [2] is a language for modeling RESTful interactions between two or more participants. Additionally, this language is situated as a middle ground between the business perspective of the business process choreography and the implementation perspective of RESTful APIs involved in the interaction. RESTful choreography language itself constitutes an extension of BPMN choreography [3] with REST-specific information.

However, RESTful choreographies lack a formal specification, thus not providing criteria for validating its correctness. To overcome this incompleteness, this paper introduces a formal specification of the RESTful choreography and two properties that each choreography diagram should satisfy for being considered complete. This allows developers to automatically check the validity of a RESTful choreography before using it, e.g., as an skeleton for the development of RESTful APIs.

The rest of the paper is structured as follows. In Sect. 2, RESTful choreographies are briefly explained and a running example is introduced. Section 3 present the reader with related work. Section 4 introduces the formal specification of business process choreography and the RESTful choreography as an extension of the former. Subsequently, Sect. 5 describes two properties of RESTful choreography, which, if satisfied, render it complete. In Sect. 6, we provide an evaluation on the usefulness of the formalizations and the completeness properties. Lastly, we conclude our paper and provide insights about future work in Sect. 7.

A. Bozzon et al. (Eds.): ICWE 2016, LNCS 9671, pp. 365–372, 2016.
DOI: 10.1007/978-3-319-38791-8_21

## 2 Foundations

RESTful choreography diagram is introduced in [2] with the aim of bridging the conceptual gap between the BPMN business process choreography [3] and its implementation as a RESTful interaction [1]. It is an enhancement of the BPMN choreography diagram with REST-specific annotations.

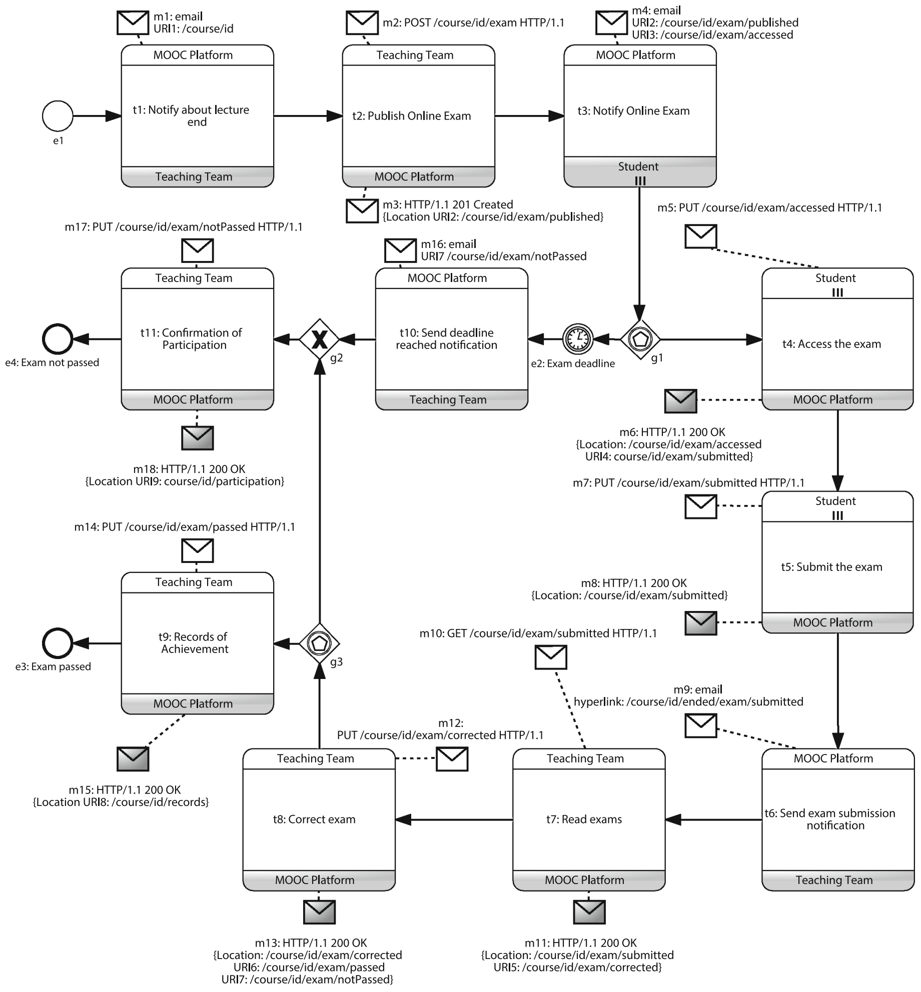
m1: email
URI1: /course/id
MOOC Platform
t1: Notify about lecture end
Teaching Team
e1
m2: POST /course/id/exam HTTP/1.1
Teaching Team
t2: Publish Online Exam
MOOC Platform
m3: HTTP/1.1 201 Created
{Location URI2: /course/id/exam/published}
m4: email
URI2: /course/id/exam/published
URI3: /course/id/exam/accessed
MOOC Platform
t3: Notify Online Exam
Student
m5: PUT /course/id/exam/accessed HTTP/1.1
m17: PUT /course/id/exam/notPassed HTTP/1.1
Teaching Team
t11: Confirmation of Participation
MOOC Platform
e4: Exam not passed
m18: HTTP/1.1 200 OK
{Location URI9: course/id/participation}
g2
m16: email
URI7 /course/id/exam/notPassed
MOOC Platform
t10: Send deadline reached notification
Teaching Team
e2: Exam deadline
g1
Student
t4: Access the exam
MOOC Platform
m6: HTTP/1.1 200 OK
{Location: /course/id/exam/accessed
URI4: course/id/exam/submitted}
m7: PUT /course/id/exam/submitted HTTP/1.1
Student
t5: Submit the exam
MOOC Platform
m8: HTTP/1.1 200 OK
{Location: /course/id/exam/submitted}
m14: PUT /course/id/exam/passed HTTP/1.1
Teaching Team
t9: Records of Achievement
MOOC Platform
e3: Exam passed
g3
m15: HTTP/1.1 200 OK
{Location URI8: /course/id/records}
m10: GET /course/id/exam/submitted HTTP/1.1
m9: email
hyperlink: /course/id/ended/exam/submitted
m12:
PUT /course/id/exam/corrected HTTP/1.1
Teaching Team
t8: Correct exam
MOOC Platform
m13: HTTP/1.1 200 OK
{Location: /course/id/exam/corrected
URI6: /course/id/exam/passed
URI7: /course/id/exam/notPassed}
Teaching Team
t7: Read exams
MOOC Platform
m11: HTTP/1.1 200 OK
{Location: /course/id/exam/submitted
URI5: /course/id/exam/corrected}
MOOC Platform
t6: Send exam submission notification
Teaching Team

**Fig. 1.** RESTful choreography for massive open online course

To motivate our approach we introduce a RESTful choreography diagram (see Fig. 1) that models an example of a massive open online course (MOOC) inspired by openHPI (https://open.hpi.de/). We focus only on the online examination procedure taking place after all lectures are published.

The main participants are the teaching team, MOOC platform, and the students. The teaching team is responsible for publishing the exam and correcting the exams submitted by the students. The MOOC platform is a system which 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 from the latter. 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 access a *Confirmation of Participation*.

As it can be observed from Fig. 1, the RESTful choreography is a business process choreography with additional REST information. The REST information is embedded in the message exchanged between participants. Each message can represent a REST request or response, or an email in case of server to client updates, e.g., MOOC platform informs the teaching team after an exam is submitted by the student.

## 3 Related Work

As a language which resides between the business processes collaboration and its platform-specific application, RESTful choreography can be compared to BPEL4Chor [4]. BPEL4Chor is an extension of BPEL [5] with choreography-related concepts. It differs from RESTful choreography because it is based on WSDL [6] and SOAP [7].

A similar work is introduced in [8]. The authors use UML sequence diagrams [9] to model REST conversation. However, their approach is limited to the modeling of the interactions between a single client and a single REST API. While, the benefit of our approach is the unlimited restriction of the number of participants and the global view on common REST resources addressed by several participants.

The same limitation holds for the RESTful conversation proposed in [10] i.e. it models the REST-based interaction between one client and one server. Despite being an extension of BPMN choreography diagram, the RESTful conversation differs from our approach in the modeling goal because it omits the textual description of the choreography task. Hence, it focuses only on the REST-based interaction and not in the business logic behind it.

## 4 Formal RESTful Choreography

In this section we introduce a formal definition of business process choreography, over which the RESTful choreography diagram is subsequently defined. The formalization of business process choreography is not a complete 1 to 1 mapping of the BPMN choreography specification [3] (specified by means of a metamodel) 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 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 finite set of nodes.* $T$ *is a non-empty, finite set of choreography tasks.* $E$ *is finite set of events. And,* $G$ *is a finite set of gateways. The sets* $T, E, G \subseteq N$ *are all pairwise mutually disjoint.*
- $S \subseteq N \times N$ *is a set of sequence flows.*
- $P$ *is a set of participants.*
- $M$ *is a set of messages.*
- $etype : E \mapsto \{start, intermediate, end\}$ *assigns an event type to each of the events of the choreography.*
- $gtype : G \mapsto \{xor, ebased, or, and\}$ *assigns a gateway type to each of the gateways of the choreography.*
- $init : T \mapsto P$ *assigns a participant as initiator to each of the choreography tasks of the choreography.*
- $recip : T \mapsto P$ *assigns a participant as recipient to each of the choreography tasks of the choreography.*
- $initm : T \mapsto M$ *assigns a message as initiating to each of the choreography tasks of the choreography*
- $retm : T \mapsto M \cup nil$ *assigns a message as return to each of the choreography task of the choreography.* $nil$ *stands for no return message.*

Moreover, we can formalize the basic rule of the choreography diagram: The initiator of each task is always aware of the immediate previous interactions, hence making the choreography enforceable [11]. In order to formally express the rule of *choreography task sequencing* we need the following notations: $p_a = (n1, n2, \ldots, n_k)$ is a path if $\forall i = 1..k, (n_i, n_{i+1}) \in S$; $\bullet t = \{t' \in T \mid \exists p_a = (t', n_1 \ldots n_k, t) \wedge \forall i = 1..k, n_i \notin T\}$ is the set of all direct-preceding tasks of $t$; $T^0 = \{t \in T \mid \bullet t = \phi\}$ is the set of all choreography tasks that have no direct predecessor; $T^* = T \backslash T^0$ is the complement of set $T^0$. Using these notations, we have:

**Definition 2 (Choreography Task Sequencing).** *Given a choreography diagram* $C = (N, S, P, M, etype, gtype, init, recip, initm, retm)$ *and a participant* $p \in P$*, the basic rule of choreography task sequencing holds iff*

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

Next, we define RESTful choreography, which extends Definition 1.

**Definition 3 (RESTful Choreography).** *RESTful choreography* $C_R = (N, S, P, M, U, etype, gtype, init, recip, initm, retm, server, mtype, hyperlink)$ *is an extension of business process choreography with the following concepts:*

- *$U$ is a set of URIs.*
- *$server : P \mapsto \{0,1\}$ marks all participants that have a server role in a RESTful interaction.*
- *$mtype : M \mapsto \{req, res, email\}$ maps any message exchanged in a RESTful choreography to one of three message types: request; response; and, email. We, then, have $\forall t \in T, server\,(recip\,(t)) = 1 \Leftrightarrow mtype\,(initm\,(t)) = req \Leftrightarrow mtype\,(retm\,(t)) = res$*
- *$hyperlink : M \mapsto 2^U$ maps each message of the RESTful choreography to set of URIs, which play the role of hyperlinks that the client can use to continue the interaction with the server. We have $\forall m \in M, mtype\,(m) = req \Rightarrow |hyperlink\,(m)\,| = 1$ because a REST request is composed of a single REST verb and a single URI.*

## 5 Completeness of RESTful Choreographies

In this section we introduce two properties that render a RESTful choreography complete. First we provide the definition of completeness.

**Definition 4 (Completeness of RESTful Choreography).** *A RESTful choreography $C_R$ is said to be complete, if it is hyperlink complete and it has a correct resource behaviour.*

In a 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 her to follow in future interactions. For server to client direction, an email communication is assumed. We do not explicitly take into consideration RESTful Push Interactions [12] because they are a special case of the normal RESTful interaction, e.g., to notify the client about new updates the role of server and clients are briefly exchanged.

To represent the fundamental importance of the hyperlink, as a steering tool for guiding the RESTful choreography, we introduce the property of hyperlink completeness. 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 first occurring choreography tasks are excluded from this criteria because they have no preceding task. Hyperlink completeness also requires that all hyperlinks sent between participants are modelled in the RESTful choreography.

**Definition 5 (Hyperlink Completeness).** *RESTful choreography is hyperlink complete if a participant $p$ sends a request via URI to the server in task $t$, then for all execution paths leading to task $t$ the request URI is passed to participant $p$ embedded in a response or email message. Formally:*

$$\forall t \in T^*, server\,(recip\,(t)) = 1 \Rightarrow \forall t^0 \in T^0, \forall p_a = (t_0, .., t), \exists t' \in p_a(t^0, .., t) \mid$$
$$(init\,(t) = init\,(t') \wedge URI\,(req\,(t)) = hyperlink\,(res\,(t')))$$
$$\vee\,(init\,(t) = recip\,(t') \wedge URI\,(req\,(t)) = hyperlink\,(email\,(t')))$$

The second property is about checking the behaviour of REST resources in that whether or not the resources involved in the choreography behave as expected. Defining this property, assures the users of RESTful choreography that each REST resource does not undergo undesired behaviour. This is particularly useful in case of RESTful choreography 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 6 (Behavioural Correctness).** *A RESTful choreography is said to have a correct resource behaviour if all 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.*

Nevertheless, these conditions apply only when we use a REST request to change the state of a resource. 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 application logic of the server.

To check the behaviour of the resource, we first derive the resource behaviour, in the form of a UML state machine [9], from the RESTful choreography. The derivation is performed for every REST resource found in the choreography. Then, we check the behaviour of each resource if it is correct or not.

The derivation procedure of the resource behaviour 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 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 a state transition diagram. State naming is based on the URI of the REST request, e.g., exam is published, accessed, submitted. State changes inducted by the REST request are labeled in the state transition diagram with the corresponding request like shown in Fig. 2.

## 6 Evaluation

In this section we apply the defined properties on the choreography depicted in Fig. 1 and argue about the usefulness of these two properties.

The hyperlink-completeness property of this choreography is checked by identifying the URI used in each REST request, and checking whether or not the URI is sent to client performing the request at some point earlier in the choreography. The automation of this procedure is left as a future work. The diagram from Fig. 1 is hyperlink complete. However, during its design, checking for this property recursively proved to be beneficial because we were able to spot flaws and resolve them, e.g., $URI_3$, $URI_6$ and $URI_7$ were not included respectively in messages $m_4$, $m_{13}$ and $m_{16}$.

For checking the resource behavioural correctness of the running example, we identified one main REST resource i.e. the *exam*. Figure 2 depicts the lifecycle of the *exam* resource derived in the manner described in the previous section. Following the definition of resource behavioural correctness we can conclude from Fig. 2 that the *exam* resource has a correct behaviour because it is: created via a POST; accessed via GET and no new state is introduced; edited via PUT leading to a new state; and finally, all requests are performed after the resource is created and before it is deleted (in this example there is no DELETE request).

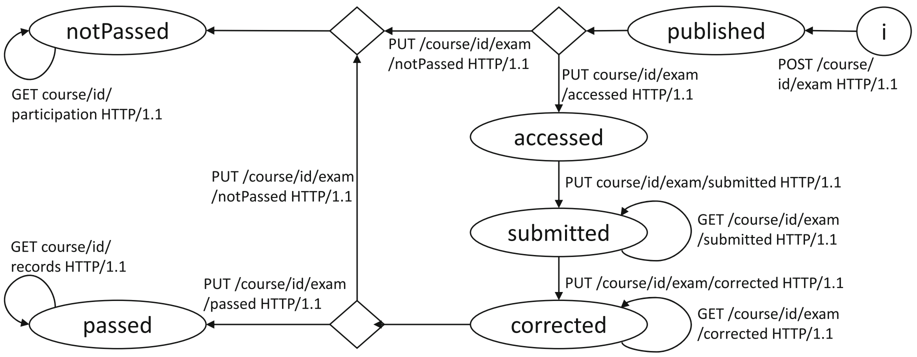
notPassed
published
i
POST /course/id/exam HTTP/1.1
PUT /course/id/exam /notPassed HTTP/1.1
PUT course/id/exam /accessed HTTP/1.1
GET course/id/participation HTTP/1.1
accessed
PUT course/id/exam/submitted HTTP/1.1
PUT /course/id/exam /notPassed HTTP/1.1
submitted
GET /course/id/exam /submitted HTTP/1.1
GET course/id/records HTTP/1.1
PUT /course/id/exam/corrected HTTP/1.1
PUT /course/id/exam /passed HTTP/1.1
passed
corrected
GET /course/id/exam /corrected HTTP/1.1

**Fig. 2.** Exam lifecycle derived from the RESTful choreography in Fig. 1

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 $/course/id/exam/notPassed$. Moreover, GET requests can be easily checked if they are safe i.e. do not introduce side effects. This is realized by making sure that every GET state transition is looped around a single state (see Fig. 2).

Since the MOOC choreography has only one main REST resource that has a correct behaviour, then the choreography has a correct resource behavior. Therefore, the MOOC RESTful choreography is considered complete because it satisfies both properties defined above.

## 7 Conclusion

In this paper we propose a formal specification of the RESTful choreography diagram together with properties for verifying its completeness. The formalization is realized by: first, formalizing the business process choreography; and second, formalizing the extension that constitutes the RESTful choreography diagram. The proposed specification is applied in an example of examination procedure in the context of a massive open online course.

The two identified properties of the RESTful choreography are hyperlink completeness and behavioural correctness. The former assures that all the hyperlinks used by the client (in the client server context) participants for making a REST request are provided to them prior to their request occurrence. The latter makes sure that all REST resources behave as expected during their lifecycle. For the second property we make use of a different view - a state transition diagram is derived from the RESTful choreography for each REST resource involved in the interaction. This additional view, provides to the user of RESTful choreography a clearer view on each resource behaviour by emphasizing the state transitions induced by REST requests. This is particularly useful in cases when REST resources are accessed and updated by different participants during the course of the interaction because it gives an overview over the allowed changes at certain points in the choreography.

**Acknowledgement.** We thank Cesare Pautasso for the valuable discussions on the BPM-REST relation, especially on the resource lifecycles.

## References

1. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis (2000)
2. Nikaj, A., Mandal, S., Pautasso, C., Weske, M.: From choreography diagrams to restful interactions. In: WESOA 2015, Co-located with ICSOC 2015. Springer, Berlin (2015)
3. OMG: Business Process Model and Notation (BPMN), Version 2.0, January 2011. http://www.omg.org/spec/BPMN/2.0/
4. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: extending BPEL for modeling choreographies. In: IEEE International Conference on Web Services (2007)
5. 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 Stand. **11**, 1–10 (2007)
6. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL) 1.1. W3c note, WWW Consortium, March 2001
7. World Wide Web Consortium: Simple Object Access Protocol (SOAP) 1.2 (2003)
8. Haupt, F., Leymann, F., Pautasso, C.: A conversation based approach for modeling REST APIs. In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA 2015), Montreal, Canada, May 2015
9. OMG: Unified Modeling Language (UML), Version 2.0, July 2005. http://www.omg.org/spec/UML/2.0/
10. Pautasso, C., Ivanchikj, A., Schreier, S.: Modeling RESTful conversations with extended BPMN choreography diagrams. In: Weyns, D., et al. (eds.) ECSA 2015. LNCS, vol. 9278, pp. 87–94. Springer, Heidelberg (2015). doi:10.1007/978-3-319-23727-5_7
11. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 2nd edn. Springer, Berlin (2012)
12. 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)