Published in Software & Systems Modeling, DOI 10.1007/s10270-016-0532-2.
(The final publication is available at: `http://link.springer.com/article/10.1007/s10270-016-0532-2`)

Visual Modeling of RESTful Conversations with RESTalk

Ana Ivanchikj · Cesare Pautasso · Silvia Schreier

Received: 15 January 2016 / Revised: 30 April 2016 / Accepted: 2 May 2016

Abstract The cost savings introduced by Web services through code re-use and integration opportunities have motivated many businesses to develop Web APIs, with ever increasing numbers opting for the REST architectural style. RESTful Web APIs are decomposed in multiple resources, which the client can manipulate through HTTP interactions with well defined semantics. Getting the resource in the desired state might require multiple client-server interactions, what we define as a RESTful conversation. RESTful conversations are dynamically guided by hypermedia controls, such as links. Thus, when deciding whether and how to use a given RESTful service, the client might not be aware of all the interactions which are necessary to achieve its goal. This is because existing documentation of RESTful APIs describe the static structure of the interface, exposing low-level HTTP details, while little attention has been given to conceptual, high-level, modeling of the dynamics of RESTful conversations. Low-level HTTP details can be abstracted from during the design phase of the API, or when deciding which API to use. We argue that, in these situations, visual models of the required client-server interactions might increase developers' efficiency and facilitate their understanding. Thus, to capture all possible interaction sequences in a given RESTful conversation, we propose RESTalk, an extension to the BPMN Choreography diagrams to render them more concise and yet sufficiently expressive in the specific REST domain. We also report on the results obtained from an exploratory survey we have conducted to assess the maturity of the field for a domain specific language, and to obtain feedback for future improvements of RESTalk.

Keywords RESTful Web Services · Conversations · BPMN Choreography · Modeling Notation Extension · Exploratory Study · Domain Specific Language · Questionnaire · RESTalk

A. Ivanchikj
USI, Lugano, Switzerland
E-mail: ana.ivanchikj@usi.ch

C. Pautasso
USI, Lugano, Switzerland
E-mail: cesare.pautasso@usi.ch

S. Schreier
innoQ Deutschland GmbH, Monheim, Germany
E-mail: silvia.schreier@innoq.com

1 Introduction

Web Application Programming Interfaces (APIs) allow systems to interact with each-other over the network, thus enabling remote access to Web services [37]. Emerging technologies, such as Cloud services [26], Service mashups [7], and Microservices [29], all make use of Web APIs, driving their improvement not only in terms of design and performance, but also in terms of usability.

There has been a recent shift in software engineering towards API-driven development, where the key focus in the development is the API [12]. Such an "API first" paradigm fosters an agile approach, not only in the service development, but also in the API development. This requires fast feedback on the API's design, which in the initial design phases can be facilitated by visualizing the designed API's behavior. Documentation of API's design can also improve the usability and learnability of the API once it is frozen and released to the public. Learning and understanding an API constitutes a major part of a programmer's work. A survey on what makes APIs hard to learn [38] has revealed that

78% of the respondents learn APIs by reading documentation. What they often find missing in such documentation, is the high-level design of API's behavior to help them understand how to use the API to achieve their goal.

Simple APIs, designed to fulfill the requirements of a single client, can minimize the number of exposed operations. Public and reusable Web APIs, however, are typically used by multiple clients, built at different times and operated by different organizations. In this case, API's size and complexity would grow out of control if an operation is added to satisfy the needs of each type of client using it. Thus, while reusable APIs publish the minimal set of operations to satisfy all clients, some clients may have to compose such operations through multiple interactions, thus having a conversation with the API to achieve their goals.

A quick look at the ProgrammableWeb website[1], a popular Web API repository, highlights the dominance of REST APIs, i.e., APIs which are compliant with the REpresentation State Transfer (REST) architectural style [10]. This has also been confirmed by a recent empirical study [40], which among other aspects, also explored how service to service communication is implemented in industry. As more and more Web services adopt the constraints of REST, conversations remain an important concept when reasoning about how clients make use of RESTful Web APIs [37] over multiple HTTP request-response interactions [16].

RESTful conversations have been introduced in [16], where they are used as an abstraction mechanism to simplify the modeling of individual RESTful APIs. In traditional messaging systems, conversations involve a set of related messages exchanged by two or more parties [18,4]. Web services borrowed the notion of conversation [5] to indicate richer forms of interactions going beyond simple message exchange patterns [45].

The existing tools for REST API documentations (e.g., RAML[2], Swagger[3], Blueprint[4], Mashape[5]) focus on structural and data modeling aspects, and as such do not capture the dynamics of the client-server interactions.

While using standard UML sequence diagrams to visually represent a sample of RESTful conversations in [16], we have realized the need for a domain specific notation. The goal of such notation would be to visualize all possible interactions that may occur in a given RESTful conversation, thus we have named it RESTalk. To design it we have decided to start from the Business Process Model and Notation (BPMN) Choreography diagrams [46, Chap. 5], since they have been designed with the purpose of modeling interactions between multiple parties. BPMN has become an ISO standard in 2013 (ISO/IEC 19510). In [33] we have extended the BPMN Choreography diagrams to emphasize relevant details for using the HTTP protocol, such as hypermedia controls [3], headers and status codes[6]. These are all important for defining the salient properties of the set of request-response messages composing a RESTful conversation.

In this article, we present RESTalk and we evaluate it with an exploratory study, whose goal is to obtain initial feedback, primarily from industry, on some of RESTalk's cognitive dimensions [13], as well as to elicit whether there is a need for explicit modeling of RESTful conversations with a domain specific language. The feedback obtained from the 35 survey participants provides a formative evaluation of our work, and will affect our next iterations with the design of RESTalk.

The promising results from the reading and modeling tasks we have assigned to the participants, are particularly encouraging for us, as is the fact that almost all participants are willing to use RESTalk in their projects, and find it equally or more understandable, efficient and concise compared to the textual or visual notation they are currently using.

The rest of the article is structured as follows. In Section 2 we define the main properties of RESTful conversations and in Section 3 we introduce RESTalk and offer some guidelines on its use. We present the design and results of the conducted exploratory survey in Section 4 and discuss them in Section 5. After studying related work in Section 6, we draw some conclusions in Section 7.

2 RESTful Conversations

REST is a hybrid architectural style [43], which combines the layered, client-server, virtual machine, and replicated repository styles, with additional constraints (i.e., the uniform interface, statelessness of interactions, caching and code-on-demand) [10]. These constraints affect the way in which the client and the server interact.

For instance, the statelessness constraint requires a client's request to carry all the relevant information for

[1] `http://www.programmableweb.com/protocol-api`
[2] `http://raml.org`
[3] `http://swagger.io`
[4] `https://apiblueprint.org`
[5] `https://www.mashape.com`

[6] This article includes and extends the work that the authors have presented at the 9th European Conference on Software Architecture (ECSA 2015) under the title "Modeling RESTful Conversations with Extended BPMN Choreography Diagrams" [33]

understanding the request, such that the server does not need to remember the state of the conversation. As a consequence, every interaction within a RESTful architecture is always initiated by the client. The client is interested in manipulating the resources hosted by the server, where resources are conceptual abstractions of any information or service that can be named, and thus can be identified as relevant to the client.

Each resource is globally identified by a Uniform Resource Identifier (URI) used to address the request to that resource. The client discovers the URIs dynamically from the server's response, which can refer the client to related resources. The mechanism whereby hyperlinks (or resource references) are embedded into resource representations, or sent along in the corresponding meta data [32], is one of the core tenets of REST, known as Hypermedia [3]. The server's responses can contain from zero to many links, depending on the current state of the requested resource. The server might also send parametrized links based on which the client can dynamically construct the URI for the next request by providing the required parameter(s). The client, by deciding which link(s) to follow, can take different paths in the conversation. However, the client should simply follow links, without making any assumptions about the URI's structure [42]. This constraint is known as HATEOAS (Hypermedia As The Engine of Application State) [10].

Another important REST constraint, that allows for decoupling, information hiding and standardisation, is the uniform interface requirement. This is achieved by using standard methods to manipulate resource representations. Which methods are available for which resource is decided at design time, but can change at run time based on the state of the resource. When using the HTTP protocol the methods may include, for example, GET, POST, PUT, DELETE. Which one will be used in the request, depends on the client's goal. Some of these methods (e.g., GET) are safe in terms that they do not modify the resource. Others (e.g.,PUT, DELETE), are not safe but they are nonetheless idempotent, i.e, they can be called multiple times without changing the outcome of the call, which has important recovery implications in presence of temporary communication failures [9].

To summarize, as a result of the above mentioned characteristics and constraints of the REST architectural style, RESTful conversations can be seen as a specific kind of message-based conversations defined by the following characteristics:

1. Interactions are always client-initiated, thus it is the client who drives the conversation forward and decides when to stop it;
2. Client requests are addressed to resources, identified through their URIs;
3. When the server is available to process client requests, every request message is always followed by a response. There may be different possible responses to the same request message, depending on the state of the requested resource;
4. Hypermedia: responses embed related URIs, which may be used to address subsequent requests;
5. Statelessness: every request is self-contained and because of that independent of the previous ones;
6. Uniform Interface: there is a fixed set of request methods a resource can support. Depending on the state of the resource, the server allows clients to use different methods when interacting with it.

These characteristics make it possible to share the responsibility for the conversation's direction between clients and servers. It is the client who initiates the conversation, but it is the server who guides the client towards the next possible steps by choosing to embed zero, one or more related URIs as hyperlinks in a response. The client may choose which hyperlink(s) to follow, if any (it may also decide to stop sending requests at any time). This way, the client decides how to continue the conversation by selecting the next request from the options provided by the server in previous responses. In general, the client can accumulate URIs discovered during the entire conversation or may remember them from previous conversations. Zuzak et al. call this the *Link Storage* in their finite-state machine model for RESTful clients [49]. Additionally, responses may be marked as cacheable, and thus clients will not need to contact the server again when reissuing the same request multiple times.

The discussion so far assumes that servers are available and always reply to client's requests. However, servers may indicate their unavailability by sending responses with the 503 Service Unavailable status code. In case of failures, either due to loss of messages or to the complete unavailability of the servers, an exception to the request-response rule must be made. Clients may thus decide to resend a request after a given timeout (for temporary failures) or eventually give up retrying (for permanent failures).

3 Visual Modeling of RESTful Conversations with RESTalk

The graphical representation of all the possible interactions, that may occur as part of a RESTful conversation, facilitates its comprehension. While UML sequence diagrams can be a good starting point when

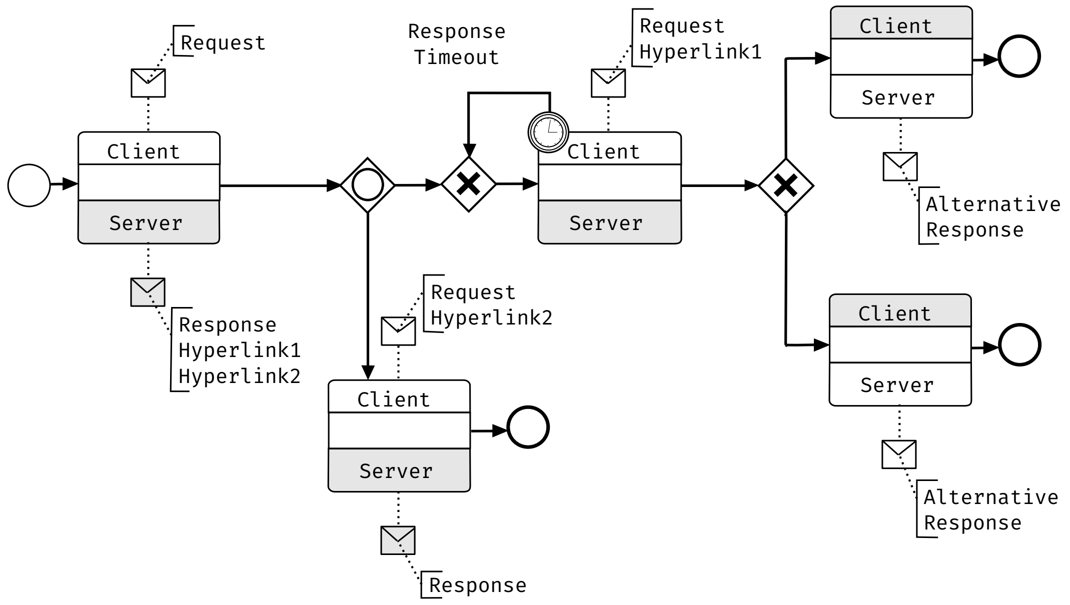

Fig. 1 Exemplary RESTful conversation modeled using standard BPMN Choreography

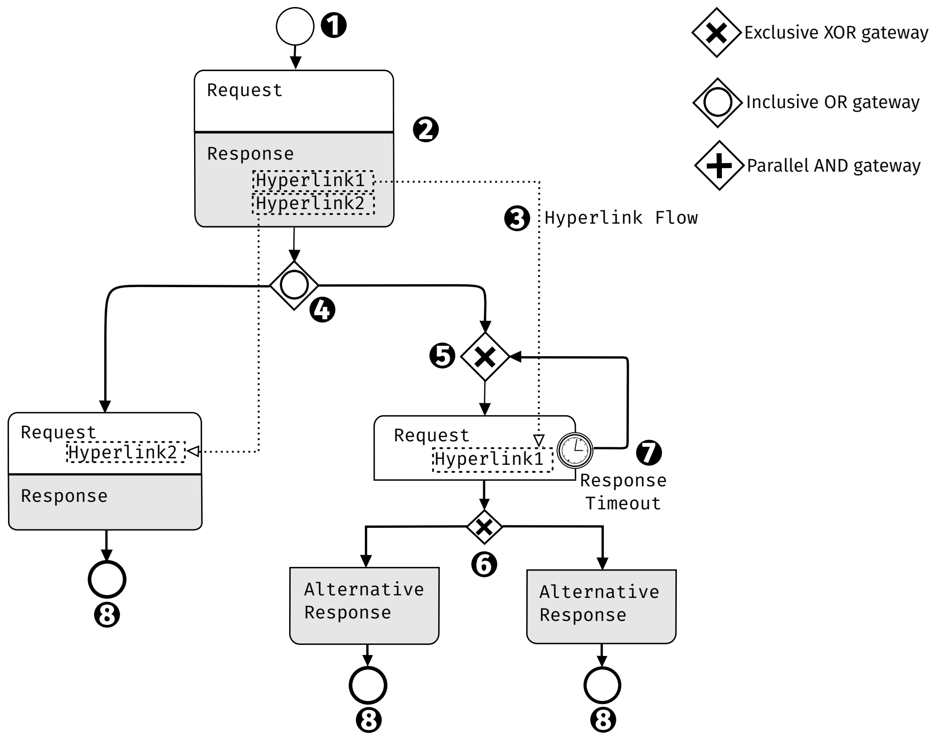

Fig. 2 Exemplary RESTful conversation modeled using RESTalk with numbered notation constructs

dealing with simple conversations [16], they are limited in concisely presenting conversations that can follow alternative paths.

3.1 RESTalk

Initially, we have considered using the BPMN Choreographies to visually model RESTful conversations [30]. They focus on the exchange of messages with the purpose of coordinating the interactions between two or more participants [21, pg. 315], and at the same time they precisely describe the partial order in which the interactions may occur. An exemplary RESTful conversation would look as in Fig. 1, if modeled using the standard BPMN Choreography notation.

However, Lindland et al., in their framework for understanding the quality in conceptual modeling [24], claim that a very important aspect of a modeling language is its domain appropriateness. Cortes-Cornax et al. [6] emphasize the same when evaluating the quality of BPMN Choreographies. They state that "the language must be powerful enough to express anything in the domain but no more".

Therefore, to render the BPMN Choreography diagrams more concise when targeting the modeling of RESTful conversations, as opposed to generic message-based conversations for which the BPMN Choreographies were originally designed, we propose minor, but significant changes to the standard notation. We name the extended notation RESTalk, and in Fig. 2 we use it to model the same exemplary RESTful conversation shown in Fig. 1. The modifications to the standard BPMN Choreography are presented below.

Modification 1: In contrast to business processes where it is important to highlight which participant is responsible for initiating the interaction, in a RESTful conversation the initiator is always the client, and there is no one-way interaction, as every successful request is followed by a response. The content of the messages is of a particular interest, because it defines, as a minimum, the resource, the action to be taken by the server, and the future direction of the conversation. To comply with these differences and bring the visual construct closer to its meaning [28], we replace the BPMN activity, comprised of an optional incoming/outgoing message with a text annotation to depict the message content and a three band choreography task containing participants' names, with a two band request-response element with embedded message content (Fig. 3). The required content of the request-response messages is the request method, the URI, as well as the response status code, and where applicable links.

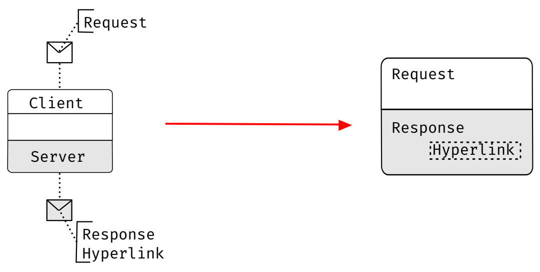

Fig. 3 Modification 1: replacing the BPMN activity

Modification 2: Since in a RESTful interaction a request is always followed by a response, the request-response bands always go together, except when there is path divergence due to different possible responses from the server to a given client's request. Only in this case the request is separated from the responses by an exclusive gateway to show the two or more alternative responses that can be sent by the server (Fig. 4).

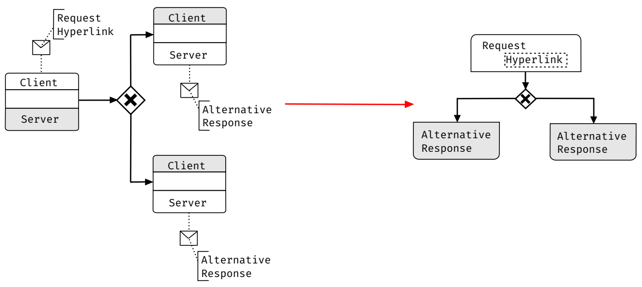

Fig. 4 Modification 2: allowing for different server responses

Modification 3: The *hyperlink flow* indicates how URIs are discovered from hyperlinks embedded in a preceding response to clarify how clients discover and navigate among related resources (Fig. 5). Adding this element is important in RESTful conversations where, due to the HATEOAS constraint in a REST compliant API, clients should not be forced to guess URIs, neither to retrieve the URIs from out-of-band knowledge [44]. The hyperlink flow makes it thus possible to distinguish which requests are sent by clients using hard-coded knowledge about URIs, and which are sent by dynamically discovering the URI from previous responses. Namely, if there is a client request in the conversation model (with the exception of the first request), where the URI is not extracted from a hyperlink flow, the API designer is aware that a client would need out-of-band knowledge to complete the conversation.

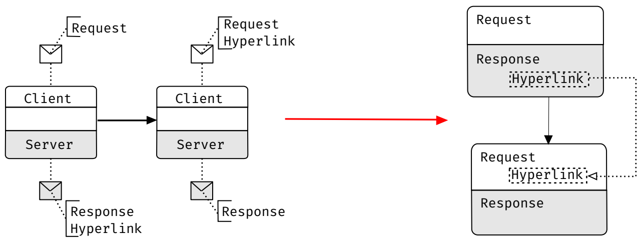

Fig. 5 Modification 3: hyperlink flow

To summarize, the core constructs in RESTalk, used to express one to one (client-server) conversations, are as following (please refer to the enumerated elements in Fig. 2 for the visual presentation which is based on the same exemplary RESTful conversation as Fig. 1):

1. Start event: to mark the beginning of the conversation;

2. An activity containing the content of the request (white) and response (gray) messages;
3. Hyperlink flow to highlight the usage of resource identifiers discovered from previous responses;
4. Control flow split gateways to show path divergence due to client's decisions, e.g., to navigate to a given resource or to end the conversation. There are three types of gateways: XOR - exclusive gateway that allows only one of the outgoing paths to be taken; OR - inclusive gateway that allows none, one or all of the outgoing paths to be taken; AND - parallel gateway that requires all outgoing paths to be taken.
5. Control flow merge gateways to show path convergence after a split. The same symbols are used as for the split gateways, but the semantical meaning is to allow the conversation to continue after a certain condition is met, i.e., for XOR - exclusive gateway the request from only one of the incoming flows has to be received; for OR - inclusive gateway the requests from all paths that have been activated with an inclusive split need to be received; for AND - parallel gateway the requests from all concurrent paths need to be received;
6. Exclusive split due to different possible responses from the server;
7. Response timeout to model situations where it is relevant for the conversation to show that, if the server takes too long to respond, the client will decide to resend the request. Such timeouts can happen after every request, but as discussed later in the simplifications, we recommend to explicitly use the response timeout event only for non-idempotent requests. This element is used attached to the request element to show its interrupting nature [21, pg.342] that breaks the normal request-response sequence, and introduces a request-timeout-request sequence. Such sequence can be repeated as long as a response is not received from the server, or the client eventually gives up;
8. End event: to mark the end of the conversation, when the client stops sending further requests. Different end events (reflecting different outcomes) are possible for a given conversation.

While many of these constructs have been formally specified in the BPMN Choreography metamodel [21, pg.316], the REST-specific metamodel (e.g., requests, responses, hyperlinks) has been formally defined by Nikaj and Weske in [31].

3.2 RESTalk Guidelines

While in the previous section we have explained the design of RESTalk, its constructs and its constraints, in this section we provide RESTalk users with certain recommendations which, if applied, can simplify the created models and make them more readable.

3.2.1 Simplifications

Often in high-level conceptual modeling [39, pg. 93], various assumptions and simplifications are necessary to avoid overwhelming the reader with too many visual elements. As a result, certain details are excluded from the model representation. Having in mind the characteristics of RESTful conversations, mentioned in Section 2, we introduce the following assumptions that help simplify the RESTalk diagrams:

1. While a hyperlink that has been discovered by the client can be used at any time in the future, to avoid decreased readability due to too many hyperlink flow edges, we only take into consideration the hyperlink obtained from the nearest previous response (Fig. 6 (S1));
2. While servers may send responses that include many different HTTP status codes, we only include the status codes which are relevant for the specific conversation. For example, 5xx status codes can occur at any time. The conversation model should only explicitly indicate how a client will need to react to such errors depending on the specific conversation domain and error semantics (Fig. 6 (S2));
3. While clients may decide to stop sending requests at any time, we model a path as finished (by using an end event), only if an initially intended goal has been achieved (Fig. 6 (S3));
4. While clients may choose to resend idempotent requests (GET, PUT, DELETE) an arbitrary number of times, we only model situations where the client retries sending non-idempotent request (POST, PATCH) after a *response timeout* event occurs. This is because resending idempotent requests does not affect the outcome of the request (Fig. 6 (S4));
5. Likewise, clients may eventually give up resending requests after a timeout. This additional branch and end event is not typically shown in optimistic models representing how clients deal with temporary failures (Fig. 6 (S5)).

The visual effect of the above mentioned simplifications is evident in Fig. 6.

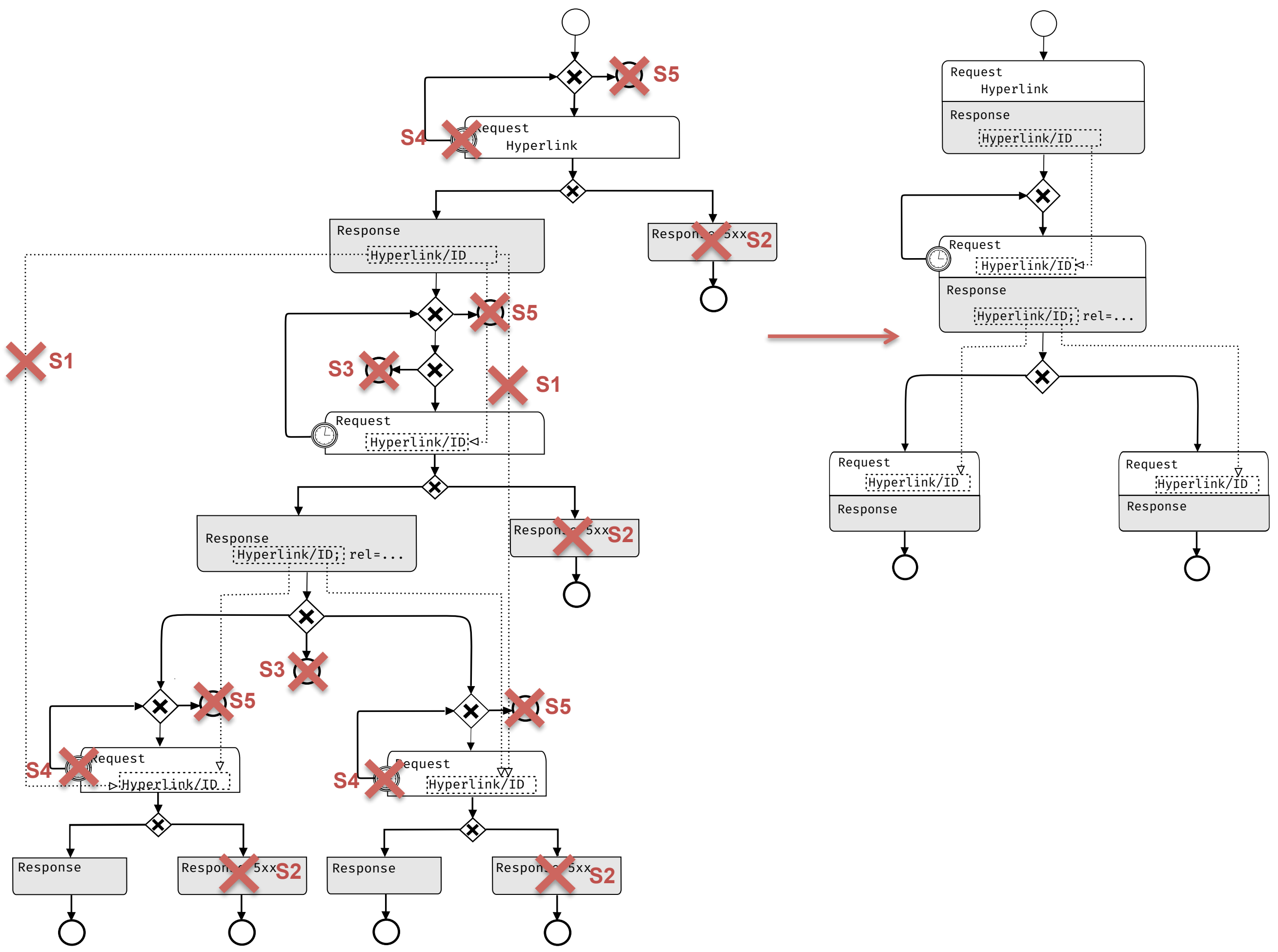

Fig. 6 RESTtalk diagram simplified following the guidelines of Section 3.2.1

3.2.2 Style Guidelines

Style is frequently influenced by the personal preferences of the modeler or by the complexity of the reality that the model needs to describe. Thus, the implementation of the following recommendations is subject to modeler's choice.

To accommodate the request-response content presented within the activities, we recommend a vertical flow for the diagram layout, as opposed to the horizontal direction usually used in standard BPMN Choreographies. This should enhance the readability of the diagram as the control flow can be followed from top to bottom: with a starting event leading directly to client's request, followed by the corresponding server's response leading directly to the next request or an end event. The events that the modeler can use are not limited to the default none or to the timer event type. BPMN Choreographies offer a plethora of event types, and depending on the needs of the conversation, any of them can be used within RESTalk. For instance, events may be used to represent the out-of-band discovery of links, e.g., when they are extracted from an e-mail message. As opposed to BPMN, where best practices advise documentation of the decisions in exclusive and inclusive gateways, in RESTalk the modeler can abstract from describing how clients make decisions on which path to follow, if such details are not entirely relevant to the conversation.

The request method, the URI, the response status code and links are mandatory elements of the request-response messages. However, RESTalk does not limit the content of a message to only the above mentioned elements. Depending on the notation usage, whiteboard or drawing tool, any headers or content details considered necessary can be added by the modeler.

Last but not least, in RESTalk, the sample values of URIs found in the hyperlink flow are meant as placeholders. They do not necessarily have to reflect the actual structure of the URI, since the server is free to create any URI to be sent to the client embedded in a response. Sometimes a concrete URI is not expressive enough, for example, for providing the client with a link for searching, where the search term typically is not known in advance by the server. In such cases, an HTML form [17] or a URI template [14] is provided by the server, so that the client can mint the concrete URI by replacing the parameter(s) in the given template. This requires the client to know the semantics of all parameters. Such URI templates can also be contained in the hyperlink flow.

4 Exploratory Survey

One general problem with modeling languages is their dissemination and acceptance by the targeted modeler community. To address this issue, we have decided to follow a user-driven approach for gradual improvement of RESTalk. Therefore, after defining the initial version and testing its expressiveness with several examples of RESTful conversations, we have conducted an exploratory survey, which is a qualitative research technique for understanding the viewpoint of the surveyed subjects about the addressed problem [47, Chap. 2]. We have opted for this approach since we wanted to inquire into the need in industry for a domain specific modeling language. Furthermore, we wanted to obtain insights to help us manage the unavoidable trade-off between the expressiveness and completeness of RESTalk on one hand, and its simplicity and understandability on the other [11].

4.1 Survey design

Given the exploratory nature of the survey, we have mostly used open-ended questions. Their purpose was to gain understanding on industry's existing practices in representing the client-server interactions, which occur when using a REST API, and to obtain respondents' opinion on RESTalk, its cognitive characteristics, and usefulness. To reach a greater audience we have translated the survey both in English and German and made it available on-line[7]. When answering all the questions in details, the expected duration of the survey as designed has been 30-40 min, which fits with the actual time spent by the participants (Fig. 7).

We have divided the questions in the following seven groups: demographic data, background on used notations in practice, RESTalk's intuitiveness, RESTalk vs. standard BPMN Choreography, reading task, modeling task, and RESTalk's evaluation.

We have started with questions about participants' background and experience in designing and using REST

[7] http://restfulconversations.limequery.com

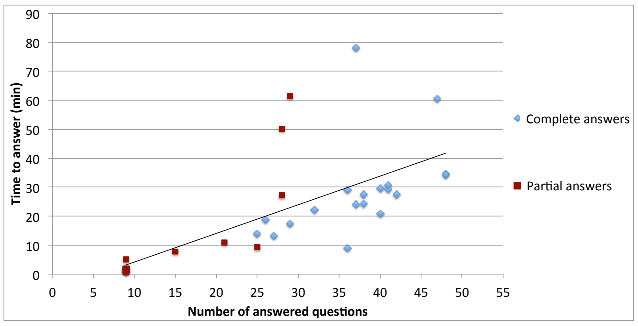

Fig. 7 Time dedicated to filling out the survey with distinction between complete and partial answers

APIs. To understand the existing modeling practices in the REST API design, we have further inquired the visual notation(s) respondents have used in such circumstances. The question concerning the used notations was a multiple-choice question in order to get a full picture of all the notations used in practice. However, the more detailed questions referred only to one of the selected notations in order to avoid constructing a too lengthy survey. The priority list for the detailed questions has been as follows: BPMN Choreography, In-house developed notation, UML Sequence diagrams, UML Activity diagrams, Other standard notations. The detailed questions were open-ended and explored respondents' perception about the pros and cons of the notation, as well as the effort of learning it and the effect its usage has had on the team's productivity.

To assess the intuitiveness of RESTalk, before explaining it in detail, we have asked the respondents to describe, in their own words, a simple conversation we had provided the diagram of, and then to respond to several specific multiple-choice questions assessing their understanding of the conversation.

As per Gemino and Wand's [11] framework for empirical evaluation of conceptual modeling techniques, such techniques ought to be compared based on their grammar, i.e., the modeling constructs and the rules for combining them. Thus, to those respondents who had basic knowledge of BPMN we have also asked some inter-grammar comparison questions. We have first explained them the extensions and the modifications we have made to the standard BPMN Choreography. Then, based on a generic example modeled in the standard BPMN Choreography (Fig. 1) and in RESTalk (Fig. 2), we have asked them to compare the two in terms of conciseness, expressiveness, and ease of understanding. We have closed this block of questions by asking which notation they would prefer using.

For the respondents who did not have prior BPMN knowledge, we have provided a short RESTalk tutorial (Fig. 2), describing all of the constructs used in subsequent tasks as well as their semantics, without diving in depth into the full complexity of BPMN. This tutorial was made available for further reference during the survey. We have not presented the assumptions and simplifications of the modeled reality mentioned in Section 3 to any of the respondents, in order to explore their validity.

To further evaluate RESTalk, we have included both a reading and a modeling task to be answered by all respondents. After these tasks, we have asked them to evaluate RESTalk in terms of conciseness, understandability and efficiency, as well as to identify any ambiguous semantics or missing elements. We have used open-ended questions in order to obtain more detailed feedback.

Not being interested in statistical inference, we have made most of the questions optional in order to encourage greater survey participation. Only the questions used in determining the survey logic, i.e., which questions to be disclosed to which audience, were mandatory. Essentially, the question regarding experience with using visual notations for modeling RESTful conversation, and the question regarding prior BPMN knowledge.

4.2 Survey sample

We have kept the on-line survey open for almost 3 months, i.e., until we have gained sufficiently valuable input from the respondents. Thus, we were not aiming at reaching a minimal sample size of the targeted population. While we were targeting primarily industry practitioners, we have also used some conferences and social media to reach a broader audience. The reason for focusing mainly on industry, was to get feedback on their willingness to use visual notations. Namely, while in academia modeling notations such as UML or BPMN are frequently taught, their acceptance in industry seems to be limited [34]. However, we believe that if a notation is developed in an agile manner, with a continuous direct contribution from industry practitioners, it is more likely to fit their actual needs and thus to achieve greater adoption.

From the total of 35 respondents, 74% (26 individual responses) are from industry and 26% (9 individual responses) are from academia, based on the job title they have provided. Their experience with using, and/or designing RESTful APIs, ranges from couple of months to more than 10 years, with almost half of them having more than 3 years of experience. More details can be found in Fig. 8. The data labels in the graph show the absolute number of respondents in each experience group. No other demographic data has been considered relevant for the current study.

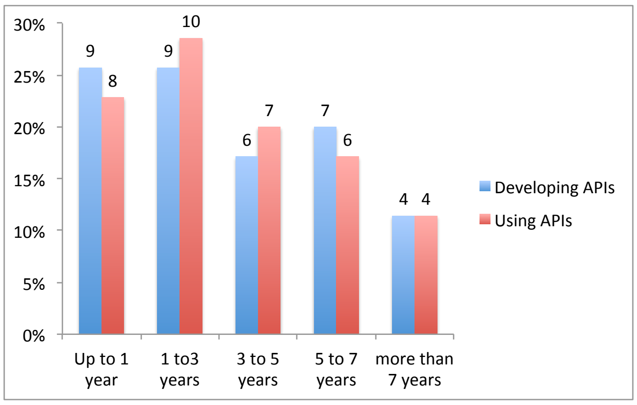

Fig. 8 Respondents' experience with REST APIs

The sample includes a broad range of practitioner roles from researchers, through IT consultants, software quality engineers, developers, architects and up to a CTO. The average time dedicated to filling out the survey has been calculated to 23 minutes, with the time increasing as the number of answered questions increased. Given the fact that the survey could be paused and resumed later, the average time has been calculated after removing the data for 3 persons who took over 3 hours to fill out the survey. Nonetheless, their answers have been taken into consideration for the rest of the survey analysis.

Since conditional logic was used in the survey, the number of questions to be answered differed among different respondents, with 8-14 additional questions for respondents with modeling experience. However, double checking the responses with time to answer values above the trend line, has indicated that they have dedicated more time to provide more exhaustive answers. Further details are available in the scatter plot in Fig. 7 where also complete vs. partial answers are evident. The scatter plot shows that all of the partial answers contain the first 10 questions, with 28 questions emerging as a limit. As can be derived from Fig. 9, this limit is due to the modeling task which the respondents probably found more challenging or time consuming.

4.3 Survey results

As mentioned in Subsection 4.1, most of the survey questions were not mandatory and survey conditional logic was incorporated in two question groups: **Used notations in practice** (Subsection 4.3.1), which was only presented to participants who have used the concrete notation before, and **RESTalk vs. Standard**

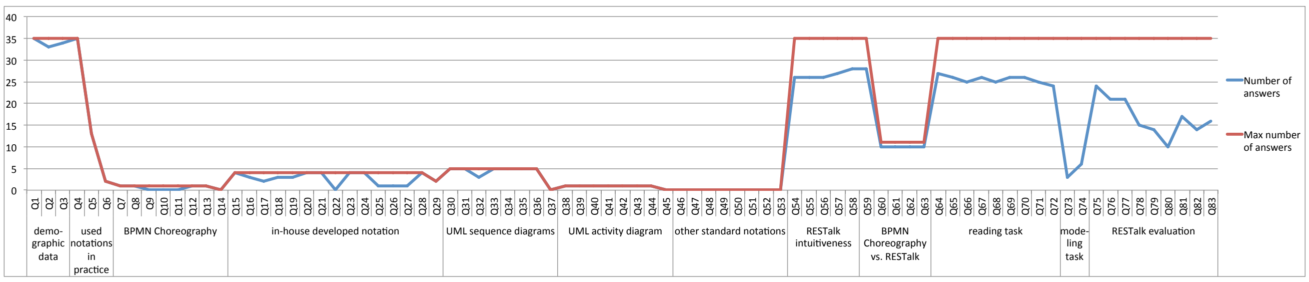

Fig. 9 Maximum vs. actual number of answers per question

BPMN Choreography (Subsection 4.3.3), which was only presented to participants with prior BPMN knowledge. Fig. 9 shows the theoretical maximum number of respondents per question, given the survey sample of 35, and the actual number of respondents. You can use it as a reference for the absolute number of respondents per question, given the percentages mentioned in the remaining part of this subsection.

4.3.1 Used notations in practice

Out of all the respondents, 38% have used some visual notations to discuss the life-cycle of resources and allowed HTTP interactions within an REST API. The percentage is equal in industry and academia. These respondents have been asked to choose one or more of the following notations they have used: BPMN Choreography, In-house developed notation, UML Sequence diagrams, UML Activity diagrams, or Other standard notations. UML Sequence diagrams have emerged as the most widely used, with as many as 85% of the respondents using it, while BPMN Choreography has been used by just one person from academia who stated that he has used it for "research incentives". Details are available in Fig. 10. Such results are not surprising given the longevity of UML. The distribution is similar even if we analyze the answers disaggregated between industry and academia.

In-house developed notations have been used by four persons, mainly due to lack of knowledge of existing standards, their complexity or lack of flexibility. These are similar reasons to what Petre has identified regarding UML use in practice in [34]. The respondents have developed simple, ad-hoc notations, or simplified the UML notation, so that the diagrams can easily be drawn by hand or with a ready-to-use tool. The one person who is no longer using the in-house developed notation, states the reason being the fact that "it was not searchable, shareable or usable after forgetting the context".

All of the persons who have declared having used UML sequence diagrams, are still using them, mainly

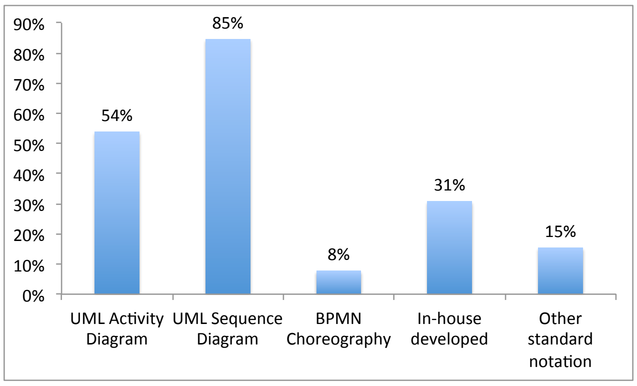

Fig. 10 Used visual notations for RESTful conversations in practice

due to their effectiveness in depicting the order of interactions between the client and the server. The UML sequence diagram language features they appreciate the most are the possibility to show the creation and destruction of resources, concurrency and the direction of the exchanged messages. Nonetheless, they consider it challenging to express "dynamic calls or a lot of third party services and failover tools", as well as conditions, loops or resource's state. Regardless of such limitations, they believe that using UML sequence diagrams has improved their productivity by improving the team's understanding of the interactions and driving design discussions. Furthermore, they believe that the UML notation has a fast learning curve, which is expected given that many study it during their formal education.

4.3.2 RESTalk's intuitiveness

A notation is intuitive if it is easy to understand without explicit instruction. Therefore, before providing any tutorial for RESTalk or explaining how it is different from the BPMN Choreography, we have shown respondents the diagram in Fig. 11, and posed an open-ended question to describe its meaning using natural language. As per the answers, it is intuitive from the diagram that an empty resource is created and the content is added later on. The concept of the hyperlink flow seems to

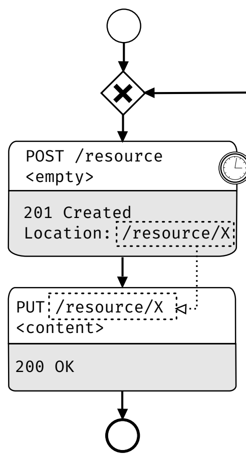

Fig. 11 Diagram used for assessing RESTalk's intuitiveness

be clear as well. The elements that have emerged as ambiguous were the exclusive merge gateway and the timer event. They depict a situation in which the server does not respond to the POST request, and thus the client sends the request again. For instance, one person expressed doubts on whether the "process makes a POST or is waiting for a POST", while another one was not sure whether the timeout occurs "if the response takes too long (or if the request is empty?)". One person from academia interpreted the gateway as an unspecified data-based decision being made.

After the open-ended question we have also asked 4 multiple choice questions with only one correct answer (marked in bold):

1. The goal of the conversation is:
   – editing an existing resource
   – **creating a new resource**
   – creating multiple new resources
   – none of the answers
2. The client can send the POST request multiple times:
   – **true**
   – false
   – I don't know
3. By sending multiple POST requests multiple resources are being created:
   – **true**
   – false
   – I don't know
4. The client knows the link to the created resource before the start of the conversation:
   – true
   – **false**
   – I don't know

As can be seen in Fig. 12, there is a fairly high average number of correct answers (77%), which we consider an indication of a rather intuitive notation. As was evident from the open-ended question answers, the timer event has caused confusion on how many resources are created if sending the POST request multiple times. While 56% have understood it well, other 22% have stated that they do not know the answer. If we disaggregate the answers by sector, the academia has higher percentage of correct answers to this question (75%). This is not surprising given that 57% of them have prior BPMN knowledge, and the timer event is one of BPMN's core constructs. It could also explain the slightly higher average number of correct answers in academia (81%) compared to industry (75%).

If we look at the intuitiveness from the respondents' experience perspective, we notice an increase in the average number of correct answers as the respondents' experience increases (Fig. 13). This is to be expected in such a simple diagram, where knowledge of the REST context can be sufficient for making educated guess on constructs which might be less intuitive. On the other hand, drilling down on the results applying as a criteria respondents' experience with using a visual notation to depict RESTful conversations, has not revealed meaningful differences in the answers between respondents who have used visual notations and those who have not.

4.3.3 RESTalk vs. Standard BPMN Choreography

Given that RESTalk is based on BPMN Choreographies, we used different approaches in explaining it to people with and without BPMN experience. As per the survey results, 41% of the total respondents have some basic BPMN knowledge, with that percentage being higher in academia (57% or 4 persons) than in industry (35% or 7 persons). After having explained the main modifications made to the standard BPMN Choreography, we have asked these respondents to compare an exemplary model designed with standard BPMN Choreography (Fig. 1) to the same model designed with RESTalk (Fig. 2). As can be seen from Fig. 14, respondents' perception is positive since nobody finds RESTalk less concise or less expressive than the standard BPMN Choreography. Only one person from industry found the standard BPMN Choreography more understandable, but without providing further details on the reasons. The number of persons who have answered the questions is provided in parenthesis in the x-axis labels in Fig. 14.

Since as we have seen in Fig. 10 UML Sequence diagrams are the most widely used notation for depicting RESTful interactions, we have decided to delve into the answers of respondents who have used UML Se-

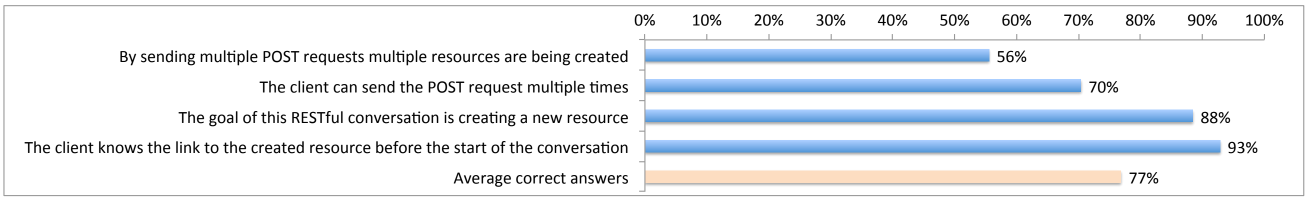

Fig. 12 Multiple choice questions for assessing RESTalk's intuitiveness (ordered by percentage of correct answers)

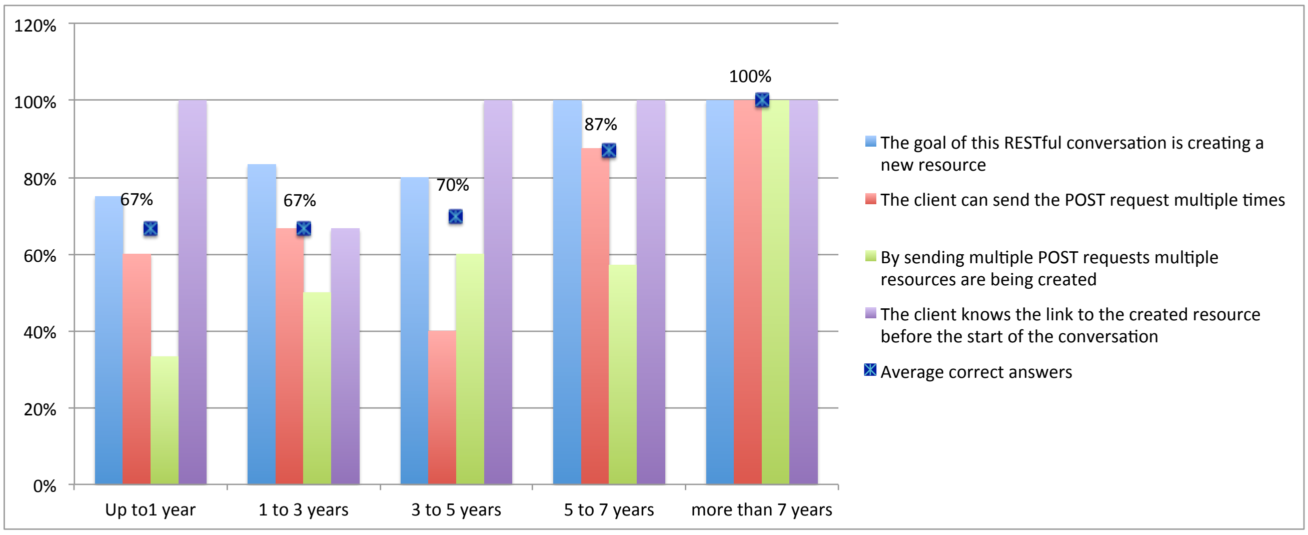

Fig. 13 Assessing RESTalk's intuitiveness from respondents' experience perspective

quence diagrams for this purpose before (7 persons). All of them but one find RESTalk less time consuming and more or equally concise than UML Sequence diagrams.

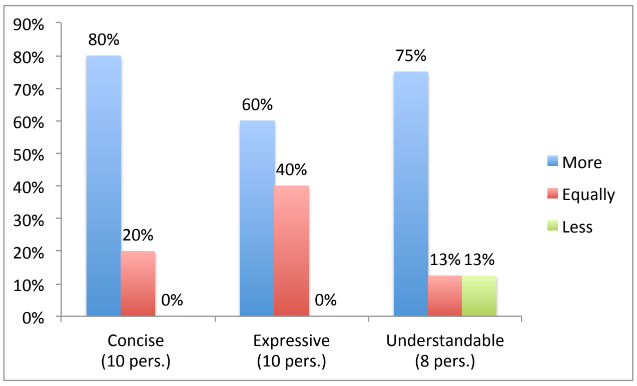

Fig. 14 RESTalk vs. Standard BPMN Choreography

If we analyze the responses comparing industry vs. academia (Fig. 15), we notice that industry participants find RESTalk significantly more expressive than the ones from academia, while they both agree on it being more understandable than the standard BPMN Choreography. These questions were answered by 4 to 6 participants from industry[8] and 4 persons from academia.

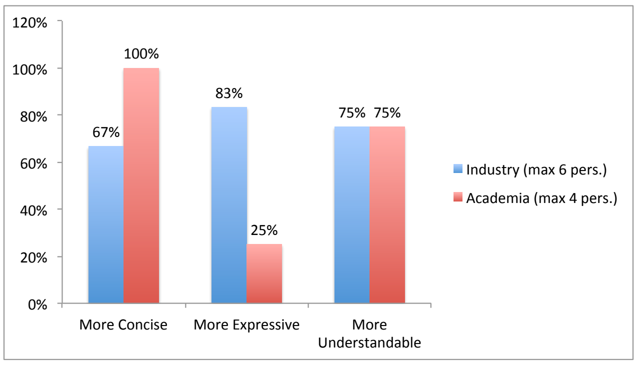

Fig. 15 RESTalk vs. Standard BPMN Choreography per sector

However, all respondents prefer using RESTalk to the standard BPMN Choreography due to lower overhead, greater simplicity and "better notion of what response belongs to what request".

[8] Since the questions were not mandatory a few industry participants did not answer all of them.

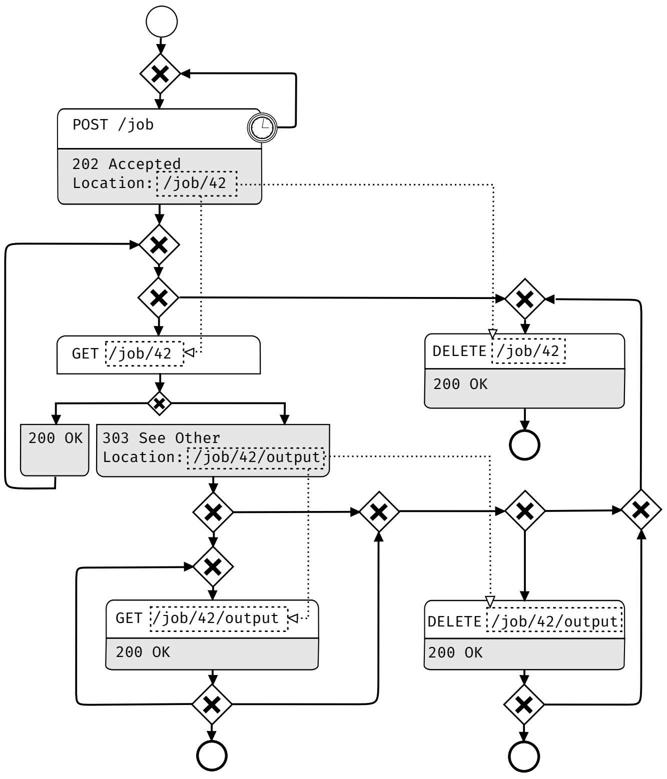

Fig. 16 Long running request conversation modeled with RESTalk

4.3.4 Reading task

For the reading task we have applied RESTalk to a common conversation that can be found in different RESTful APIs, e.g., Amazon Glacier's API for long term storage of infrequently used data[9]. Fig. 16 shows the diagram included in the survey. It depicts a RESTful conversation where the retrieval of data is turned into a job resource in order to avoid the client having to keep the connection open for too long, while waiting for the data retrieval. The client can keep on polling for the job output, and will only get redirected to it when the job has finished. Then it can read the job output multiple times, or it can decide to delete it. The job itself can be deleted at any point, thus either implicitly stopping the job, if it has not finished yet, or deleting the finished job since it is no longer necessary.

This group of questions followed after all the respondents got introduced to the basics of RESTalk. To test their ability to read diagrams modeled in RESTalk, we have posed the following multiple choice questions with one correct answer (marked in bold):

1. How many resources are created during this conversation:
   - none
   - one
   - two
   - **one or two**
   - more than two
2. What happens when you try to access the job resource while the job has not completed yet:
   - you get a 200 OK status code and a placeholder link to where the output will be saved once the job has completed
   - **you get a 200 OK status code and can try to access the job again later**
   - you cannot send a GET job request before the job has completed
3. When can you delete the job resource?
   - only after the job has completed
   - only before the job has completed
   - only after having read the output
   - only before having read the output
   - only after the job has completed and you have read the output
   - only after the job has completed, but before you have read the output
   - **at any time after the creation of the job resource**

We have also asked the following true/false/I don't know questions (the correct answer can be found in brackets):

[9] `http://docs.aws.amazon.com/amazonglacier/latest/dev/job-operations.html`

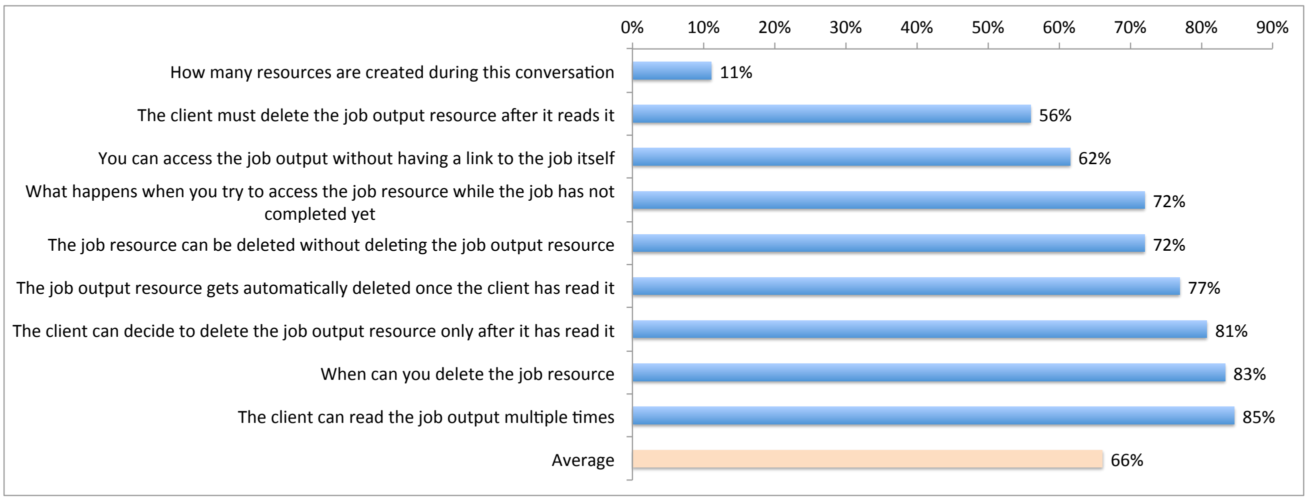

Fig. 17 Assessing the reading of RESTalk diagrams (questions are ordered by percentage of correct answers)

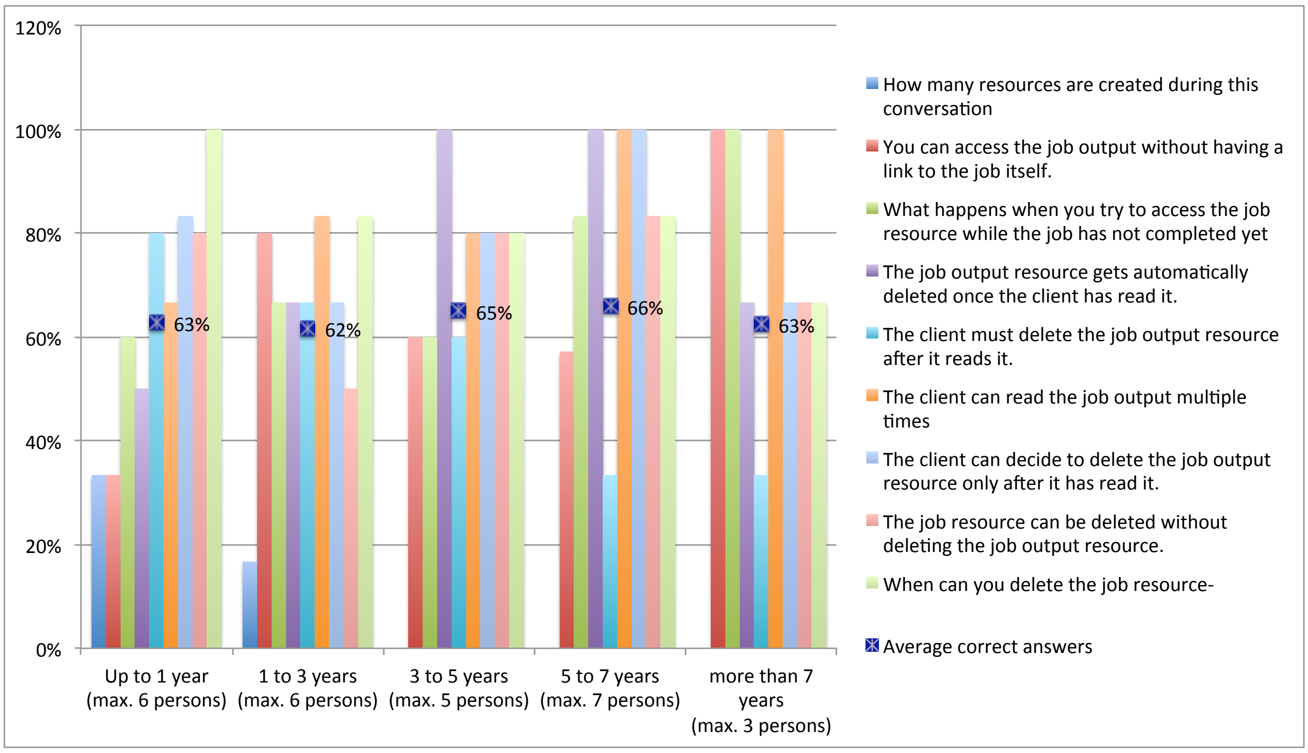

Fig. 18 Assessing the reading of RESTalk diagrams from respondents' experience perspective

1. You can access the job output without having a link to the job itself (**false**);
2. The job output resource gets automatically deleted once the client has read it (**false**);
3. The client must delete the job output resource after it reads it (**false**);
4. The client can read the job output multiple times (**true**);
5. The client can decide to delete the job output resource only after it has read it (**false**);
6. The job resource can be deleted without deleting the job output resource (**true**).

As evident from Fig. 17, all of the questions were answered with more than 50% of accuracy, except the question regarding the number of resources created during the conversation. Since the job can be deleted even before it is finished, we considered "one or two" as a correct answer, i.e., either only the job resource or both the job resource and the job/output resource are created. However, as per the discussion with some of the respondents, probably many of them (63%) consider the job/output a sub-resource of the job resource, and thus have identified "one" as the correct answer. Hence, if we regard this question as a conceptual rather than modeling question, and therefore discard it in the calculation, the average number of correct answers rises from 66% to 73%. On sector level, this indicator is much higher in academia (86%) than in industry (68%), while on BPMN knowledge level, it is higher for respondents with no previous BPMN knowledge (76%) than for respondents with basic BPMN knowledge (70%).

While in the intuitiveness group of questions, the impact of experience on the correctness of the answers was evident, this is not the case in the reading task group of questions, possibly due to the greater complexity of the modeled reality. No meaningful differences among experience groups have been noticed in this case. Fig. 18 shows the results as well as the maximum number of respondents per group.

Although the correlation coefficient between time to answer the reading task questions and accuracy of the answers is not high (0.25), a scattered plot of such correlation (Fig. 19) reveals several clusters of respondents. Those who rushed through the questions without paying too much attention (accuracy less than 40% in less than 10 minutes), those who understood well RESTalk and/or the REST concept and answered the questions quickly and accurately, and those who took their time to reason to get to the correct answers (accuracy higher than 70% in up to 50 minutes).

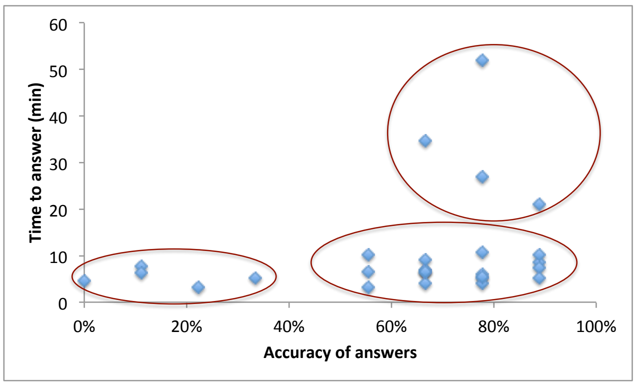

Fig. 19 Correlation between time to answer and accuracy of answers

4.3.5 Modeling task

We have asked the respondents to use RESTalk to model the life-cycle of a collection item, i.e., all the possible CRUD (Create, Read, Update, Delete) operations that can be performed on the same resource. Only three of the respondents, all from academia, have performed the task. They have all modeled the reality from the server's perspective, i.e., using just one end event after the resource is deleted. Thus, they made the simplification of only modeling with an end event a conversation which could never be resumed in the future, and abstracting from situations where the client ends the conversation without deleting the resource. They have all correctly used the activities with the request-response content, and two of them have also used the hyperlink flow element. The exclusive split and join gateways have been merged in one combined gateway by two of the respondents (Fig. 20 and Fig. 21), while the third respondent made a mistake in merging the control flows, which resulted with an infinite loop between reading and updating the resource (Fig. 22).

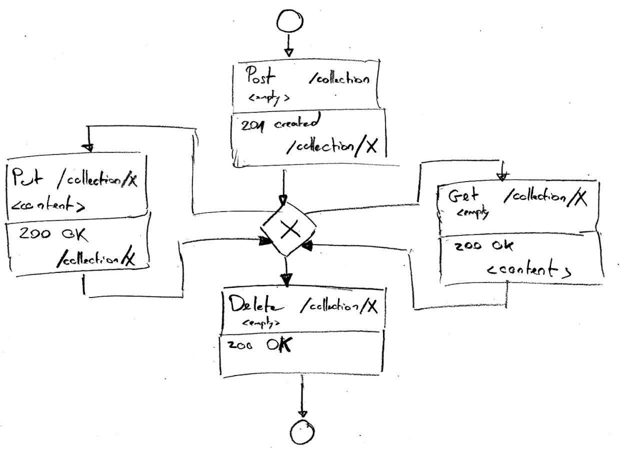

Fig. 20 Respondent's model of CRUD operations on a collection item (1)

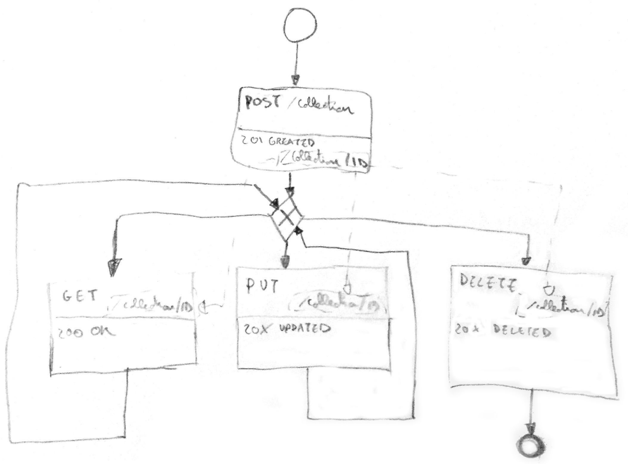

Fig. 21 Respondent's model of CRUD operations on a collection item (2)

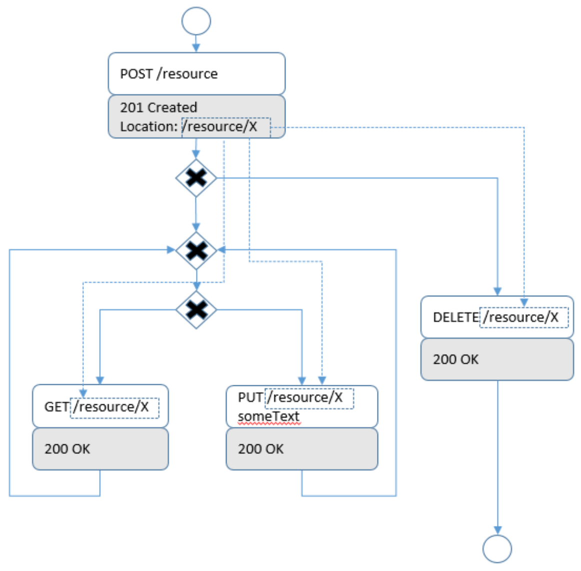

Fig. 22 Respondent's model of CRUD operations on a collection item (3)

The fact that 2 out of 3 respondents have decided not to use a tool to perform the modeling task, indicates the appropriateness of RESTalk for both whiteboard discussions and API documentation. Respondents' personal opinion on the matter is presented in Subsection 4.3.6.

4.3.6 RESTalk's evaluation

After having used RESTalk, to understand or model a RESTful conversation, we have posed open-ended questions to assess whether it is considered easily understandable, and how it stands in terms of conciseness and efficiency compared to the textual or visual notation that the respondents had used before. To give some structure to the obtained answers, we have classified them in the groups shown in the graphs (Fig. 23 to Fig. 26).

83% of the respondents have a positive view on the understandability of RESTalk, finding it easy or somewhat easy to understand. One IT consultant stated that it is "surprisingly easy". While almost all the respondents from academia find it decisively easy to understand, industry respondents seem to be more reluctant (Fig. 23). The ones that did not find RESTalk easy to understand, did not provide further details for such evaluation, while others expressed doubts of its understandability with more "complex conversations".

57% of the respondents find RESTalk less time consuming than the notation (textual or visual) they had used before (Fig. 24)[10]. All the ones who found RESTalk more time consuming are respondents who have not used visual notations before to depict RESTful conversations. Thus, it is reasonable that they consider drawing a diagram more time consuming than textual description. An IT consultant states that a "prerequisite for easy usage is that the graphical notation can be derived from a simple textual notation". As expected, given the results from the understandability assessment (Fig. 23), respondents from industry are more skeptical than in academia (Fig. 25), however the differences are not as evident as in the understandability question.

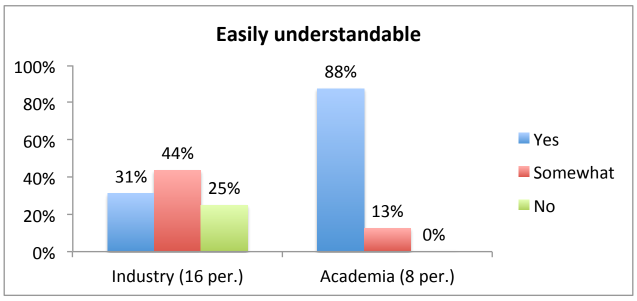

Fig. 23 Assessing RESTalk's understandability per sector

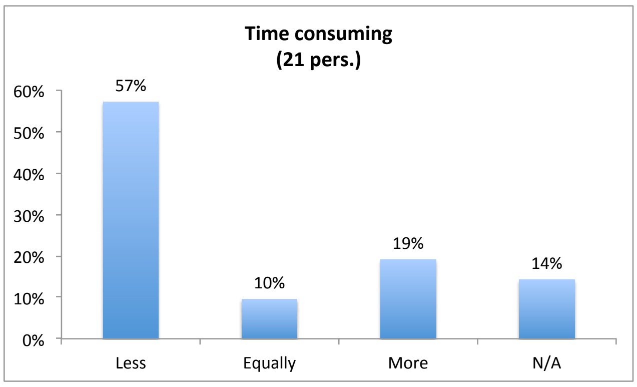

Fig. 24 Assessing RESTalk's efficiency

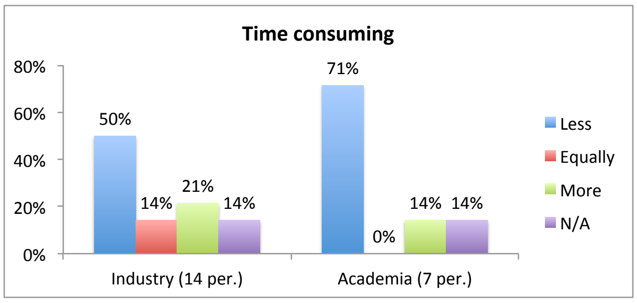

Fig. 25 Assessing RESTalk's efficiency per sector

[10] N/A stands for persons who did not use any notation before and thus could not make the comparison

Of those 18 who responded to the conciseness question, 61% consider RESTalk more concise. One respondent states that it "probably depends on the graph and the number of responses at a given request". When analyzing this indicator from the sector viewpoint, contrary to the previous results, in this case it is the industry respondents who are more united in their positive judgement about RESTalk (Fig. 26).

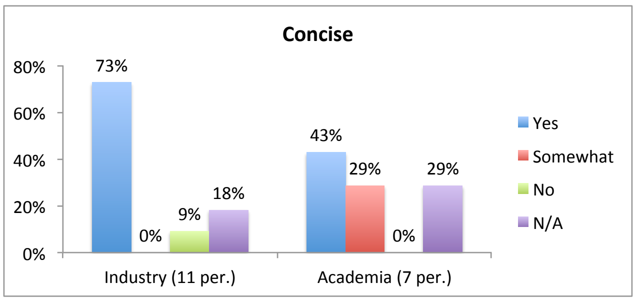

Fig. 26 Assessing RESTalk's conciseness per sector

If we define positive sentiment about RESTalk as an opinion that RESTalk is superior or equal to existing notations, then a rather high percentage of support is present in all three indicators (understandability - 83%, efficiency - 67%, and conciseness - 72%).

Regardless of the above disclosed assessment, all respondents but two are willing to try using RESTalk in their own projects. One freelance software engineer states: "it will take some time to get familiar with but I think we could boost the development process even more". An IT consultant, on the other hand, conditions its acceptance to the availability of a specific tool: "If I could simply generate it from a textual description of the graphs, I would use it". 69% of the respondents seem to have preference for a modeling tool support for RESTalk to facilitate the drawing, although some do agree that simple diagrams can also be drawn manually on whiteboards or using existing drawing tools. One also proposes to develop a tool for automatic extraction of the diagrams from code.

When asked whether they find the HTTP details (methods, URIs, status codes, links) sufficient for understanding what the RESTful conversation is aiming at, 71% have responded affirmatively. Some of the proposed HTTP elements to be added include authentication details as well as "some additional headers and parameters".

Interesting remarks have emerged from open-ended questions on RESTalk missing elements or ambiguities, as well as RESTful conversations used in practice which the respondents find challenging or impossible to model with RESTalk. Although the BPMN modeling best practice guidelines [2, Chap. 2] advise not to use combined gateways (one gateway as both a split and a join), several respondents have stated that multiple gateways create unnecessary "visual and mental distractions". In BPMN, exclusive gateways are data based decisions, so some considered it confusing not having the precise data conditions defined in the gateway. Some have also emphasized that RESTalk lacks details regarding the length of the timer or the maximum number of iterations allowed in a loop. Also people who deal with applications where the state changes are crucial, have noticed that the state transition is not explicitly marked in the current version of RESTalk. Another constructive remark is that currently it is not clear when there is a dependence between resources (e.g., a resource and a sub-resource). This is an important structural aspect, since updating or deleting the resource might result with the automatic update or deletion of the sub-resource.

5 Discussion

The purpose of this empirical study was to obtain the first cycle of feedback on RESTalk, the modeling notation we are proposing, in order to guide further improvements. Although 35 respondents might not seem high as an absolute number, we consider the constructive remarks we have obtained as truly valuable for our future work.

5.1 Need for a Domain Specific Language

Despite of the fact that currently there is no domain specific notation for modeling dynamic aspects of REST APIs, such as RESTful conversations, 38% of the respondents have already used some existing UML standard notation, or an in-house developed notation, to depict sequences of client-server interactions. As a CTO indicates, a notation is "helpful for explaining concrete scenarios. It permits a 'behind the scenes look' and thereby understanding of consequences". This indicates that there is an identified need for visualizing RESTful conversations in practice. However, even the currently most widely used notation, the UML Sequence diagram, has its limitation when it comes to depicting RESTful interactions. One programmer names the following challenges when using UML Sequence diagrams: "If you want to know more about an object than just if it's engaged (active) in an interaction or not, this can be difficult to represent on a sequence diagram and it can make the diagram too complex and unreadable. Another case when a sequence diagram can become too complex very quickly, is when we have multiple objects and object lifelines that we wish to represent, and the interactions between the objects are too convoluted to read". Our goal with proposing RESTalk as a domain specific notation is exactly to overcome such limitations of the existing standards.

5.2 Evaluation of the Cognitive Characteristics of RESTalk

Since this was an exploratory survey, with no intention of making statistical inference based on the same, we have only tackled some of the cognitive dimensions defined in [13], which we considered the most important in this initial stage of RESTalk's design. Namely, we have addressed the consistency, the closeness of mapping and the abstraction gradient dimensions.

We have taken intuitiveness as a broader approach to what is considered consistency in [13], i.e., "When some of the language has been learnt, how much of the rest can be inferred?". Without providing any prior information about RESTalk, it was encouraging to see that most of the respondents understood great part of the presented diagram and answered correctly to 77% of the multiple choice questions. In some cases, we have realized that prior knowledge of BPMN causes certain degree of confusion, particularly in circumstances where, due to the REST architecture domain characteristics, there is a slight difference in the semantics between the standard BPMN Choreography and RESTalk. For instance, while in BPMN Choreography exclusive gateways are data based and explicitly showing the decision point options is necessary, in RESTful interactions client's conditional decisions can be abstracted from. However, the respondents' feedback has made us realize that including the rationale behind server's decisions could indeed facilitate understanding under which condition each alternative response can be expected. It is also encouraging to see that even respondents with no prior BPMN knowledge have high average score of 76% of correct answers on the reading task.

The results of what is defined in [13] as a "closeness of mapping" between the notation and the problem world, are both positive and beneficial for future improvements. Namely, when asked to compare RESTalk to the BPMN Choreography, all of the respondents have found it at least equally, if not more concise and expressive. On the other hand, when asked to compare it to the notation they are currently using only one person found it less concise, while 61% found it more concise than what they are currently using. This is an encouraging result, since it involves the first iteration of the design of RESTalk. However, what has been particularly beneficial for us is that the respondents have provided several examples of conversations which cannot be modeled with the existing elements in RESTalk. For instance, conversations where the state transitions are important for the conversation logic, and should thus become explicitly visualized. Likewise, in some cases the number of maximum retries for accessing a temporary unavailable resource are limited and need to be explicitly specified. Thus, in the future, we can work on finding a way to effectively deal with such situations.

Another important cognitive characteristic of a notation, which we were interested in, was the "abstraction gradient", i.e., the levels of abstraction. While, as stated in Section 3, we have abstracted from some details when designing RESTalk, we have intentionally omitted to state the assumptions and simplifications behind such abstractions in the tutorial provided in the survey. We wanted to evaluate whether some abstractions are excessive, and thus respondents would identify them as missing elements or vice-versa. The modeling task, for instance, has revealed that the respondents have taken the end event simplification even further. While we abstract from modeling an end event after every server response, which is the real world description, they have opted for only modeling one end event to show that the conversation can never be continued in the future. On the contrary, we have used end events to show all the potentially successful conversation logs that can occur in a given conversation. The reading task, on the other hand, has disclosed that RESTalk lacks the expressiveness in showing the dependencies between resources, or that the length of the timer is not specified. Regardless of such findings, the understandability of RESTalk reaches a fairly high level of 66% based on the correct answers of the reading task multiple-choice questions, and an even higher level based on the respondents' subjective opinion, where 83% have evaluated RESTalk as easy or somewhat easy to understand.

To conclude, the predominantly positive sentiment regarding general understandability, conciseness and efficiency of RESTalk per se, or in comparison to standard BPMN Choreography or other visual notations, and above all the willingness of 88% of the respondents to put RESTalk into practice, gave us the assurance we have sought to continue improving RESTalk and dealing with some of the modeling challenges it poses.

In our future work we will need to improve the design of RESTalk to respond to some of the provided feedback, either by adding new elements or by introducing what is defined as a secondary notation, i.e., "use of layout, colours, other cues to convey extra meaning" [13] or visual variables (e.g., size, orientation, texture etc.) [28]. We are aware that there will always be trade-offs between different cognitive dimensions during the design of RESTalk. However, the iterative, user-centric approach we are following should help us find the sweet spot. After all, we do not propagate using RESTalk as the sole technique for documenting REST APIs. It merely focuses on the API's dynamic behaviour thus leaving out static details. They can be added to RESTalk as textual annotations or notes disclosed on click when using a modeling tool. Or depending on modeler's goal, RESTalk can be used as a complementary technique to already existing techniques, which are more inclined to static details.

6 Related Work

The related work ranges from structural modeling over state-based modeling approaches up to tools for API design.

Schreier [41] has identified structural (static) and behavioral (dynamic) modeling of REST APIs when defining a REST metamodel. The static model defines the structural elements of an API such as the resources, their URI, the methods that can be called on them, the supported representation media types etc. Significant theoretical work [10,25,15] and tool support (e.g., RAML, Swagger, Blueprint, Mashape) has already been provided for the structural aspect. The behavior model refers to the request-response interactions and behavior triggered by method calls. Existing work on the behavioral aspect usually addresses the question of validating compliance with REST. As such it tends to rely on Petri Nets [22] or UML state machines [41,36] to depict the dynamics.

Li and Chou [22] have used Coloured Petri Nets [20], what they call REST Charts, to model REST APIs as a set of hypermedia representations and transitions between them. Later on in [23], they have extended their modeling framework in order to decouple the resource representation from the resource connections and to provide for layered representations. The interaction itself is depicted in the transition element, however it lacks REST specific visual presentation, i.e., the request and response are not evident in the transition element. Rauf et al., tackling the research question of designing REST compliant and dependable Web services in a PhD thesis [36], have identified the necessity of modeling the behavior of REST interfaces. They have opted for using UML class diagrams and UML protocol state machines to visualize the behavioral interfaces in order to take advantage of existing tools for model validation and consistency analysis. While they have focused on RESTful service composition, our scope is wider and refers to RESTful conversations which can refer both to composed and single services. Their model is state centered and focuses on any data sent with POST, PUT or DELETE requests in order to trigger the transfer between states. The GET method is implicitly used to check for the state invariants. Based on Schreier's metamodel [41], van Porten in [35] has developed and evaluated a visual notation for modeling Resource-oriented applications. Beside views for statical aspects, he has offered two views for dealing with the behavioural aspects of the model. One focuses on the behavior of single method calls, thus not tackling multiple client-server interactions. The other view focuses on the state transitions of a single given resource.

As opposed to the previously mentioned approaches, our model has the interaction (request-response) as a first class citizen, and focuses on the hyperlink discovery through navigation. Alarcon and Wilde [1] have also embraced the important REST principle of interlinked resources and proposed ReLL (the Resource Linking Language) for describing REST services. They agree that the most important aspects in describing REST services are the links between resources and the necessary interactions to access the resources. The tool, they have implemented to harvest REST resources, outputs a typed graph of the discovered resources and the links between them. RESTalk is more detailed and, in addition to these elements, it explicitly presents the request method, server's responses with status codes and the control flow. Depending on the targeted user and the characteristics of the API, different notations can be used to visualize REST APIs. Mitra [27], for example, proposes a sketching tool aimed at Web API designers, which offers two different canvas views depending on the API style, which he calls CRUD and Hypermedia style. The CRUD style reflects the resource, URI and HTTP method, while the Hypermedia style uses informal state diagram for visualization. This tool, by design, does not support logical behavior visualization. RESTalk, on the other hand, does not only target API designers, but also API users. It aims at supporting them in achieving their goals and speeding up their learning curve by showing all the possible conversations they can have with the service. Thus we consider some of the existing work (e.g., [36,27]) as complementary to ours, offering a different perspective on the REST API visualization. While other approaches choose to focus mainly on visualizing state transitions, we have decided to offer a new viewpoint and focus on interactions.

The necessity of modeling Web service interactions is as old as Web services themselves. It has led to the creation of the "Web Services Choreography Working

Group"[11] in 2002 which aimed at defining a vendor-neutral choreography specification to facilitate service integration. The Web Services Choreography Description Language (WS-CDL) remained only as a candidate standard language since the working group was closed in 2009. However, graphical representation of the choreographies was never in the scope of this group. Nonetheless, meanwhile academia and industry were working on that aspect of choreographies and came up with different modeling language proposals such as Let's Dance [48] or iBPMN [8]. They have eventually led to the introduction of the Choreography Diagram in version 2.0 of the BPMN standard [46, Chap. 5], which Nikaj et al. [30,31] have used to model RESTful conversations by adding REST-specific annotations. However, since the main targeted domain of these languages is modeling interactions involving traditional Web services, their capability of depicting effectively and efficiently RESTful interactions is limited, which has motivated our work on visually modifying and extending the BPMN Choreography.

7 Conclusion and Future Work

Conversations are relevant in the context of RESTful Web APIs, because multiple basic HTTP interactions are combined by clients navigating through the API's resources guided by the hyperlinks provided by the server. Thus, the design of RESTful APIs always consists of conversations and not only, for example, of the URI patterns and supported media types of its resources. We consider giving a visual representation of RESTful conversations an important first step towards understanding and improving how RESTful APIs are designed, documented, and used. This has motivated our work on extending the standard BPMN Choreography diagrams to create RESTalk, which has a domain specific focus on the facets of RESTful APIs, e.g., hyperlink flow, request-response sequencing [33].

We believe that the doctrine of agile development should not be limited only to software. The design of modeling notations also needs frequent iteration, driven by user feedback. Therefore, after the initial informal feedback during the European Conference on Software Architecture (ECSA 2015) [33], we have decided to conduct an exploratory survey, as part of our strategy for gradual improvement of RESTalk. The survey was conducted among both designers of RESTful APIs and developers of client applications consuming them. Thus, the main contribution of this article consists of the insight into how well, and to which extent the existing real world practices of modeling RESTful conversations can be supported. The results of the survey have confirmed the maturity of the need for a domain specific modeling language, and the potential interest in industry for using such language. They have also provided us with a good level of confidence in the intuitiveness, expressiveness and understandability of RESTalk, the domain specific modeling language we propose.

The obtained feedback regarding missing or ambiguous constructs, as well as real world problems which would be difficult to model with the current version of RESTalk, will be used to improve the language in the future. Taking into consideration the current trends of Web service composition and mash-ups, in the future we will also extend RESTalk to express multi-party conversations. We have already started working on some of these issues and we present our initial ideas in a poster paper [19], where we apply the notation to model the Doodle API[12]. After finalizing such modifications of the language, in line with our iterative development strategy, we will conduct further surveys and controlled experiments which will focus both on RESTalk and on its comparison to other notations. The goal is to have as a final product a domain specific language that helps RESTful API designers and users to increase their efficiency and improve the quality of their work. We consider it important to assess the potential for RESTalk's acceptance in practice before deciding whether tool support should be provided for it.

On the other hand, based on our experience, existing literature and informal feedback gathered from the survey respondents, we plan to use RESTalk to visually model a collection of frequently used RESTful conversation patterns and to explore how to compose together reusable patterns to simplify the modeling of larger conversations. As Haupt et al. [15] who propose composite resources to support effective modeling, we also believe that complex RESTful conversations can be modelled with the help of interconnected set of simple named RESTful conversation patterns.

Acknowledgements We would like to thank all of the participants in the survey and the anonymous reviewers for their time and valuable feedback.

[11] `http://www.w3.org/2005/12/wscwg-charter.html`

[12] `http://support.doodle.com/customer/en/portal/articles/664212-doodle-wizard-api`

References

1. Alarcon, R., Wilde, E.: Linking Data from RESTful Services. In: Third Workshop on Linked Data on the Web. Raleigh, North Carolina (2010)
2. Allweyer, T.: BPMN 2.0: Introduction to the Standard for Business Process Modeling. BoD–Books on Demand (2010)
3. Amundsen, M.: Building Hypermedia APIs with HTML5 and Node. O'Reilly (2011)
4. Barros, A., Dumas, M., ter Hofstede, A.H.: Service Interaction Patterns. In: Business Process Management, *LNCS*, vol. 3649, pp. 302–318. Springer (2005)
5. Benatallah, B., Casati, F., et al.: Web service conversation modeling: A cornerstone for e-business automation. Internet Computing, IEEE **8**(1), 46–54 (2004)
6. Cortes-Cornax, M., Dupuy-Chessa, S., Rieu, D., Dumas, M.: Evaluating choreographies in BPMN 2.0 using an extended quality framework. In: Business Process Model and Notation, *LNBIP*, vol. 95, pp. 103–117. Springer (2011)
7. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer (2014)
8. Decker, G., Barros, A.: Interaction Modeling Using BPMN. In: Business Process Management Workshops, *LNCS*, vol. 4928, pp. 208–219. Springer (2008)
9. Fielding, R., Reschke, J.: Hypertext Transfer Protocol–HTTP/1.1. Request for Comments: 7230 (2014). URL `https://tools.ietf.org/html/rfc7230`
10. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
11. Gemino, A., Wand, Y.: A framework for empirical evaluation of conceptual modeling techniques. Requirements Engineering **9**(4), 248–260 (2004)
12. Goteti, H.: API Driven Development, Bridging the gap between Providers and Consumers. Tech. rep., CA Technologies (2015). URL `http://rewrite.ca.com/us/articles/application-economy/apis-bridging-the-gap-between-providers-and-consumers.html`
13. Green, T.R.G., Petre, M.: Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. Journal of Visual Languages & Computing **7**(2), 131–174 (1996)
14. Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: URI Template. Request for Comments: 6570 (2012). URL `https://tools.ietf.org/html/rfc6570`
15. Haupt, F., Karastoyanova, D., Leymann, F., Schroth, B.: A model-driven approach for REST compliant services. In: International Conference on Web Services (ICWS 2014), pp. 129–136. IEEE (2014)
16. Haupt, F., Leymann, F., Pautasso, C.: A conversation based approach for modeling REST APIs. In: Proc. of the 12th WICSA 2015. Montreal, Canada (2015)
17. Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., OConnor, E., Pfeiffer, S.: HTML5. A vocabulary and associated APIs for HTML and XHTML. W3C Recommendation (2014). URL `http://www.w3.org/TR/html5/forms.html`
18. Hohpe, G.: Let's have a conversation. Internet Computing, IEEE **11**(3), 78–81 (2007)
19. Ivanchikj, A.: RESTful conversation with RESTalk -the use case of doodle-. In: Proceedings of the International Conference on Web Engineering (ICWE'16), pp. 583–587. Springer (2016)
20. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 1. Springer Science & Business Media (2013)
21. Jordan, D., Evdemon, J.: Business Process Model And Notation (BPMN) Version 2.0. OMG (2011). URL `http://www.omg.org/spec/BPMN/2.0/`
22. Li, L., Chou, W.: Design and Describe REST API without Violating REST: A Petri Net Based Approach. In: Web Services (ICWS), 2011 IEEE International Conference on, pp. 508–515 (2011)
23. Li, L., Chou, W.: Designing Large Scale REST APIs Based on REST Chart. In: Web Services (ICWS), 2015 IEEE International Conference on, pp. 631–638 (2015)
24. Lindland, O., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. Software, IEEE **11**(2), 42–49 (1994)
25. Masse, M.: REST API Design Rulebook. O'Reilly (2011)
26. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (2011)
27. Mitra, R.: Rapido: A Sketching Tool for Web API Designers. In: Proceedings of the 24th International Conference on World Wide Web, WWW '15 Companion, pp. 1509–1514. Florence, Italy (2015)
28. Moody, D.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Trans. Softw. Eng. **35**(6), 756–779 (2009)
29. Newman, S.: Building Microservices. O'Reilly (2015)
30. Nikaj, A., Mandal, S., Pautasso, C., Weske, M.: From Choreography Diagrams to RESTful Interactions. In: The Eleventh International Workshop on Engineering Service-Oriented Applications, pp. 3–14 (2015)
31. Nikaj, A., Weske, M.: Formal Specification of RESTful Choreography Properties. In: Proceedings of the International Conference on Web Engineering (ICWE'16), pp. 365–372. Springer (2016)
32. Nottingham, M.: Web Linking. Request for Comments: 5988 (2010). URL `https://tools.ietf.org/html/rfc5988`
33. Pautasso, C., Ivanchikj, A., Schreier, S.: Modeling RESTful Conversations with Extended BPMN Choreography Diagrams. In: D. Weyns, R. Mirandola, I. Crnkovic (eds.) European Conference on Software Architecture, LNCS, pp. 87–94. Springer (2015)
34. Petre, M.: UML in practice. In: Proceedings of the 2013 International Conference on Software Engineering (ICSE), pp. 722–731. IEEE Press (2013)
35. van Porten, O.: Development and Evaluation of a Graphical Notation for Modelling Resource-Oriented Applications. Master's thesis, FernUniversität, Hagen, Germany (2012)
36. Rauf, I.: Design and validation of stateful composite RESTful web services. Ph.D. thesis, Turku Centre for Computer Science (2014)
37. Richardson, L., Amundsen, M., Ruby, S.: RESTful Web APIs. O'Reilly (2013)
38. Robillard, M.P.: What makes APIs hard to learn? Answers from developers. Software, IEEE **26**(6), 27–34 (2009)
39. Robinson, S., Brooks, R., Kotiadis, K., Van Der Zee, D.J.: Conceptual Modeling for Discrete-Event Simulation. CRC Press, Inc. (2010)
40. Schermann, G., Cito, J., Leitner, P.: All the services large and micro: Revisiting industrial practices in services computing. Tech. rep., PeerJ PrePrints (2015)
41. Schreier, S.: Modeling RESTful applications. In: Proceedings of the Second International Workshop on RESTful Design, pp. 15–21. ACM (2011)

42. Steiner, T., Algermissen, J.: Fulfilling the Hypermedia Constraint Via HTTP OPTIONS, the HTTP Vocabulary In RDF, And Link Headers. In: Proceedings of the Second International Workshop on RESTful Design, pp. 11–14. ACM (2011)
43. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley (2009)
44. Verborgh, R., Hausenblas, M., Steiner, T., Mannens, E., Van de Walle, R.: Distributed Affordance: An Open-World Assumption for Hypermedia. In: Proceedings of the 4th International Workshop on RESTful Design, pp. 1399–1406. ACM (2013)
45. Völter, M., Kircher, M., Zdun, U.: Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware. Wiley (2013)
46. Weske, M.: Business Process Management: Concepts, Languages, and Architectures, 2nd edn. Springer (2012)
47. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer (2012)
48. Zaha, J.M., Barros, A., et al.: Let's dance: A language for service behavior modeling. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, pp. 145–162. Springer (2006)
49. Zuzak, I., Budiselic, I., Delac, G.: A Finite-State Machine Approach for Modeling and Analyzing RESTful Systems. Journal of Web Engineering **10**(4), 353–390 (2011)