BPMN for REST

Cesare Pautasso

Faculty of Informatics
University of Lugano (USI)
via Buffi 13, CH-6900 Lugano, Switzerland
http://www.pautasso.info/
c.pautasso@ieee.org

Abstract. The *Representational State Transfer (REST)* architectural style has seen substantial growth and adoption for the design of modern Resource-Oriented Architectures. However, the impact of fundamental constraints such as stateful resources, stateless interactions, and the uniform interface have had only limited uptake and impact in the *Business Process Modeling (BPM)* community in general, and in the standardization activities revolving around the BPMN notation. In this paper we propose a simple and minimal extension of the BPMN 2.0 notation to provide first-class support for the concept of resource. We show several examples of how the extended notation can be used to externalize the state of a process as a resource, as well as to describe process-driven composition of resources.

1 Introduction

Whereas the BPMN notation has been originally developed for modeling complex message-based interactions between process-backed services [3], in the last few years a novel abstraction has emerged (the resource [5]) which changes some of the assumptions and gives new constraints for the design of service oriented architectures [21]. Since business process modeling is one of the foundations for service reuse and composition [9], it becomes important to study how modeling techniques and notations developed for message-based service choreography and orchestration can also be applied to resource-based (or RESTful [16]) Web services.

In this paper, we take a look at the general problem of how to combine *Business Process Modeling (BPM)* with the REpresentational State Transfer (REST [5]) architectural style within the specific context of the BPMN notation [11]. The goal is to study how well the basic modeling concepts and constructs of the BPMN notation fit with the resource abstraction and to propose a lightweight, minimalistic and simple extension to fill the current gap between BPMN and REST. The proposed extension aims at reusing the existing graphical elements of BPMN as much as possible in order to avoid to further increase its visual complexity [10]. With it, it becomes possible to model so-called RESTful business processes, which can be both used to orchestrate and compose a set

R. Dijkman, J. Hofstetter, and J. Koehler (Eds.): BPMN 2011, LNBIP 95, pp. 74–87, 2011.

of distributed and independent resources, as well as to give a high level representation of the behaviour of stateful resources [12]. Whereas some BPMN engines[1] are starting to feature experimental support for REST, with an HTTP-based API to deploy, manage and execute processes, current solutions are still experimental and incomplete. Thanks to our approach, modelers can use BPMN for REST to give an explicit representation of the resources involved in the business process at design-time and precisely control which process elements should be published through a RESTful Web service API at run-time.

The rest of this paper is organized as follows. In Section 2 we give some background on REST and discuss why BPMN as-is should be extended to natively support RESTful Web services. In Section 3 we give an informal definition of the proposed extensions to the BPMN, while in Section 4 we show how these extensions can be used to model three non-trivial RESTful business processes. Related work is cited in Section 5, before we draw some conclusions in Section 6 and outline some future work in Section 7.

2 Background and Motivation

The REST architectural style [4] was introduced to give a principled design of the architecture of the World Wide Web and to explain its quality attributes (loose coupling, scalability, resilience to long-term change, intrinsic interoperability). Whereas most of these quality attributes are also shared by service-oriented architectures, it is under debate whether the corresponding message-based, publish-subcribe technologies are fully capable of enabling them [18]. In the past few years, REST (or more precisely its underlying HTTP protocol [1,19]) was rediscovered and proposed as an alternative way to approach the design of such service-oriented architectures, which – according to some – could be renamed as resource-oriented architectures [16].

More in detail, a RESTful Web service publishes to its clients a number of resources, which are globally addressable by means of URIs. Clients employ a stateless communication protocol to access the uniform interface associated with each resource. The uniform interface defines a standard and common set of verbs (or methods) which can be performed on a resource. Resources are dynamically discovered by means of decentralized referral and can have multiple representations, which can be negotiated by clients. These three basic constraints (resource identification, uniform interface and multiple representations) are usually visualized as the "REST triangle", which we use (Figure 1) as an inspiration for the resource icon used in the BPMN for REST extension.

Due to the emphasis placed on the reliable transfer of state between clients and resources, RESTful services significantly differs from the basic service abstraction supported by WS-* [6,20]. For example, the WS-Resource Framework [7]) provides an additional layer of complexity by extending the SOAP protocol to build stateful services. While it borrows the notion of resources from REST, it

[1] `http://www.activiti.org/userguide/index.html#N12156`

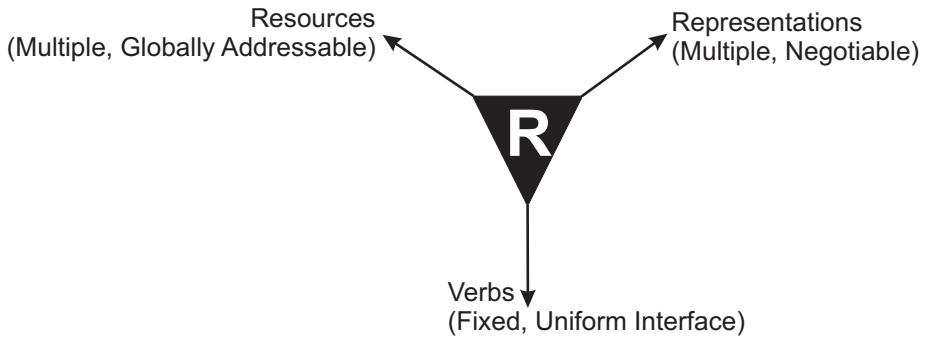
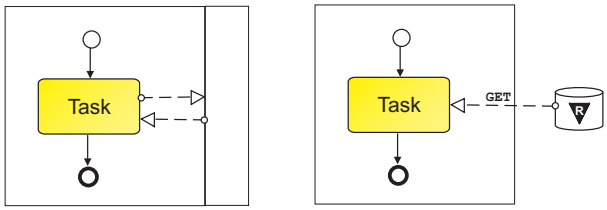


Fig. 1. The REST triangle and the Resource symbol

fails to provide a simple solution for the corresponding notions of uniform interface and global addressability of resources. Thus, it is still remains a challenge to completely wrap the resource abstraction within a traditional service-based interface [15,14]. As a consequence, there is some mismatch between the REST architectural style (originally developed for the design of distributed and decentralized hypermedia systems) and SOA technologies (such as BPEL/BPMN) addressing the needs for describing distributed systems built out of the integration and reuse of interoperable services.

In this paper we are concerned with one particular mismatch, which concerns the reuse of services by means of process-based composition, orchestration and choreography. This was originally achieved with the BPEL standard [13] and today with the latest version of the BPMN standard. These languages and the corresponding notations make a strong assumption about the set of composition techniques [2] that are available to compose services. For example, the languages provide specific constructs for message-based interaction (e.g., send/receive message icons which can be associated with tasks and events, as well as message-flow edges which literally visualize the communication between different processes). Whereas it is possible to attempt to mimick the synchronous client/server interactions of the HTTP protocol using the same elements of the notation (Figure 2 a) — after all SOAP originally proposed as a transport-independent envelope format that can wrap arbitrary content in XML so that it can be sent in terms of messages over any kind of transport protocol [17] — we believe that a more abstract and expressive notation is needed to represent at a higher level of abstraction the semantics of such interactions (Figure 2 b). Following the spirit of BPMN, the challenge consists of hiding such low-level communication details of the HTTP protocol, which are the concern of technical people. On the contrary, the goal is to let the business process modeler focus on capturing the more fundamental RESTful interactions at a higher level of abstraction without having to specify them as exchanges of HTTP request/response messages.

In particular, it should be possible to use a BPMN model to answer the following modeling questions:



(a) BPMN 2.0 (b) BPMN for REST

Fig. 2. Modeling a task which performs a read-only GET request on a resource using BPMN (a) and BPMN for REST (b)

- Which are the resources that a process depends upon for a successful execution?
- Which are the resources that are affected by the execution of a process?
- Can we reason about the behavior of stateful resources using a process model?
- Which are the tasks of a process that have been made accessible to clients as a resource?
- Which are the possible requests that can be sent to a resource whose behavior is specified by a process?

The goal of this paper is to start a discussion on a possible minimal extension to BPMN to be able to answer these and other similar questions. We will use several examples in Section 4 to show that there is indeed a benefit of directly and natively including in a BPMN model information suitable to answer these modeling questions.

3 Notation

The BPMN for REST extension augments a small number of BPMN stencils with the resource icon (Figure 3 a). The goal is to keep the extension as minimal and lightweight as possible in order to avoid adding too many new graphical symbols to an already complex notation. In order to avoid confusing the resource icon shape with the existing signal shape, which also uses a triangle, the orientation

(a) (b) (c) (d)

Fig. 3. The resource icon (a) applied to tasks published as a resource (b) and the new resource request event (c) which should be distinguished from the standard signal event (d)

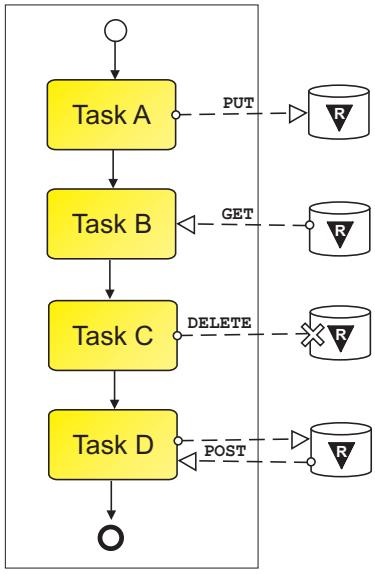


Fig. 4. Interaction with external resources

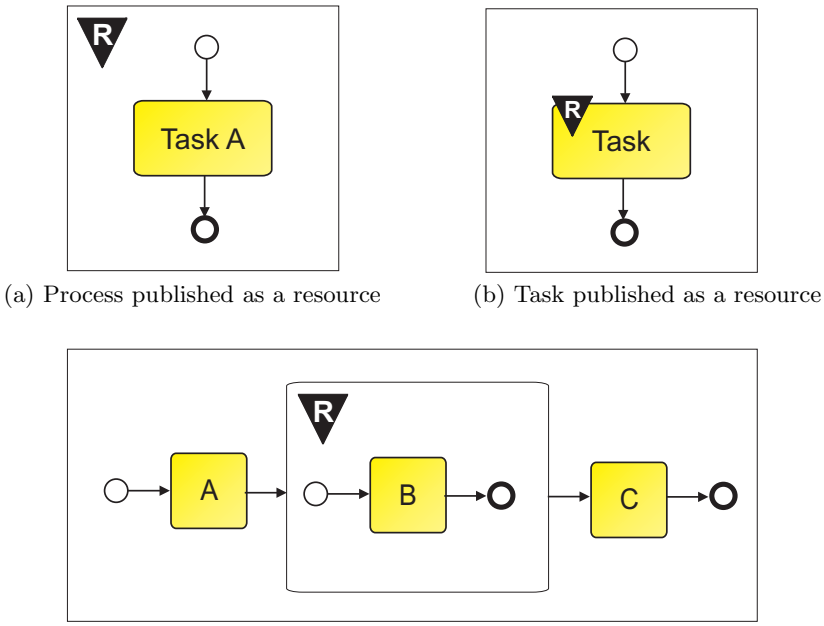
of the triangle used for the resource icon has been reversed (compare Figure 3 c vs. d).

In this section we describe the notation for modeling the interaction between processes and external resources (i.e., resource orchestration). Additionally, we also show two different ways for processes to publish resources and handle requests directed to their resources.

3.1 Modeling External Resources

Resources whose lifecycle is independent of a specific processe instance (e.g., a remote RESTful Web service API) are represented using the data store symbol refined with the resource icon (Figure 4). The intention is to depict an external place where processes can read or write persistent data, with the additional constraint that this place is a resource: the data store is addressed with a unique identifier (a URI which is not shown in the diagram, but kept as a property attribute of the shape) and it is accessed using the uniform interface (which is more expressive than basic read/write operations as in the current BPMN standard).

To specify the kind of interactions of tasks with such external resources we suggest to use the message flow edges (since the intention is to model information flowing across organizational boundaries) and to annotate the edges with the actual verb to be invoked on the resource's uniform interface. This way a precise model of the interaction can be visualized. In particular, the direction of the



(a) Process published as a resource

(b) Task published as a resource

(c) Sub-process published as a resource

Fig. 5. Publishing processes, tasks and sub-processes as resources

message flow edge reflects the information flow associated with the method[2]. GET requests fetch information from the resource into the task consuming it; PUT requests are symmetric since they allow tasks to update the state of a resource, thus the information flows from the task into the external resource; POST requests enable bi-directional information flow (thus the double message flow edges). DELETE requests do not model any message flow, thus the edge head shape has been slightly modified to visualize the "destructive" effect on the resource targeted by the request.

3.2 Publishing Process Elements as Resources

Publishing process elements as resources entails defining a mapping between the resource abstraction and some BPMN constructs so that modelers can declaratively specify which process elements should be published as a resource. To do so, we propose to visually tag with the resource icon the processes, tasks, or sub-processes which should correspond to a resource (Figure 5). The lifecycle of such resource, as opposed to an external resource, will be implicitly entangled with the lifecycle of the process instance.

[2] Whereas in the notation examples used throughout the paper we chose to include specific HTTP methods, the notation would work in a similar way for resources having methods defined as part of a different, non-HTTP-based uniform interface.

More specifically, processes published as a resource get their own URI (e.g., `/{process}`) and follow a predefined behavior when handling requests through the resource uniform interface. Processes which are not published as a resource use an implementation-specific mechanism for their execution, which may or not involve the use of a RESTful Web service interface. The `/{process}` resource acts as a "resource factory" [1] as it allows clients to initiate the execution of new process instances by sending POST requests to it. Following the POST-REDIRECT-GET pattern[3], the client will receive an identifier of the newly started process instance (e.g., (`POST /{process}; 302 Redirect, Location: /{process}/{instance}`)) and the execution of the process will continue in the background. The clients may then use such identifier to safely retrieve (with `GET /{process}/{instance}`) the result of the process once it completes its execution. Process instances may still be created using the other mechanisms (message events, receive tasks, event-based gateways) foreseen by BPMN. Also in this case process instances get their own URI, which however has to be discovered by clients through a channel which should be independent of the instantiation mechanism (e.g., by asking the process engine to enumerate the URIs associated with the process instances of a certain user).

At any time, clients can also use the process instance resource identifier to retrieve a global view over a running process instance by GETting its representation, which – depending on the chosen media type – may contain links to the individual tasks which have been published as a resource. A client may be interested in only listing all active tasks of a process instance, as opposed to retrieving links to all tasks and then having to poll each task to determine its state. By default task URIs can be automatically generated by concatenating the process instance resource idenfier with the name of the task BPMN element (i.e., (`/{process}/{instance}/{task}`). It may be possible to override such default naming convention with a manually defined URI associated with the task. As shown in [8], a more complex URI template would be necessary to distinguish multi-instance tasks (in case those are published as resources). Concerning tasks which are found within loops, the URI would point to the most recent state of the loop, i.e., so that clients can bookmark and retrieve a representation of the state associated with the task most recent iteration.

Once a task URI has been retrieved, a client may perform a GET request on it to read task-specific information (e.g., its state, its input/output parameter values). Clients may also perform a PUT request to change the state of a task (i.e., to indicate that its execution has completed) and set the value of its output parameters. Clients are not allowed either to POST or DELETE individual task resources. Once all tasks have completed their execution, their final state remains associated with the corresponding resources until a `DELETE /{process}/{instance}` request is performed. Only then, all information associated with all tasks of a process instance is removed.

[3] This could also be implemented using the 201 Created status code, which however is not yet fully supported by Web browsers, which will not continue the navigation to the URI found in the Location header unless the 302 Redirect code is used instead.

As shown in Figure 5 (c), it is also possible to associate resources with subprocesses. The idea is that these resources become visible to clients only during the execution of the sub-process. Once the execution leaves the sub-process block, then the resources are not longer visible. Clients may perform GET requests on the corresponding identifier (`/{process}/{instance}/{sub-process}`) to retrieve the state of the resource associated with the sub-process. PUT requests can also be allowed so that information from clients can flow into the sub-process and affect the behaviour of the tasks found within. In general, POST and DELETE are not allowed. In fact, DELETE requests could be used to allow clients to trigger the cancellation of the sub-process block (assuming that the corresponding cancellation handlers have been attached to the sub-process block).

Whereas the details of how to map processes/tasks to resources can be further refined, the main goal is to abstract the complexity of the interactions here described and very simply depict the difference between private tasks of the process model from tasks that become accessible from clients through a predefined RESTful Web service interface.

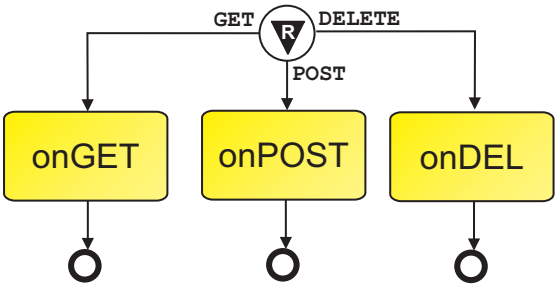


Fig. 6. Handling different request methods (e.g., GET, POST, DELETE) with the resource request event. Methods which are not explicitly modeled (e.g., PUT) will result in a `205 method not allowed` response status code.

3.3 Modeling Internal Resources: The Resource Request Event

For a more detailed and fine-grained model of how processes can be used to specify what happens inside resources, we propose to introduce a new kind of top-level event (Figure 6). This event is triggered whenever a client performs a request on the corresponding resource. The event can discriminate the different verbs associated with the request so that different tasks of the process can be activated depending on whether a GET or a POST request was received by the resource. Graphically, we associate the verbs with the control flow edges outgoing from the event. For simplicity and consistency the resource request event reuses the same resource icon as before.

The execution semantics of the resource request event is analogous to existing BPMN events. A process may accept a request sent to a resource even if the execution path triggered by previous requests has not yet completed. Multiple

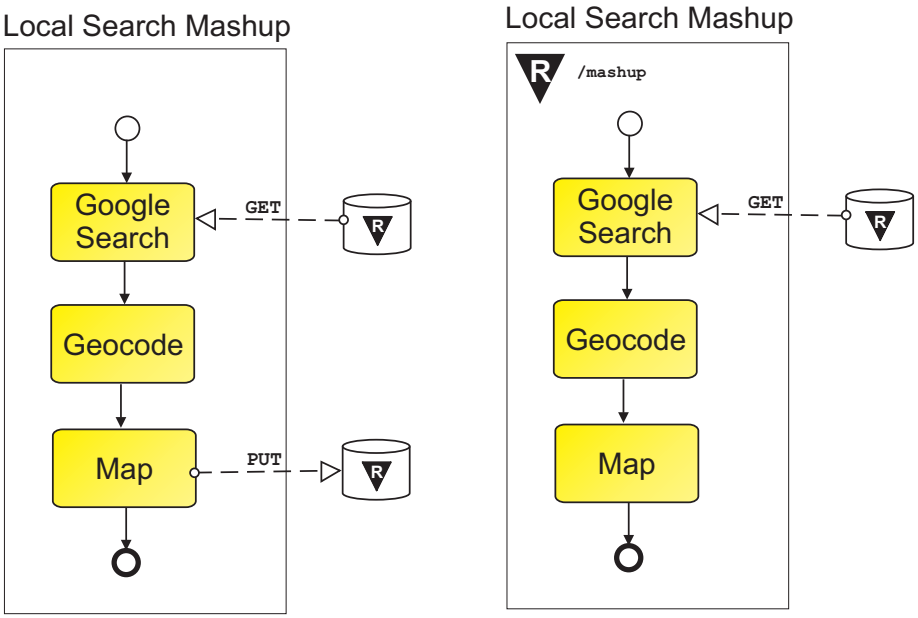
execution of the tasks associated with the resource request event can be serialized for POST, PUT, DELETE requests, while read-only safe GET requests may be executed concurrently for optimization purposes. It is important that the tasks associated with the request handling paths of a request event conform to the safety and idempotency properties of the corresponding methods.

4 Examples

4.1 Local Search Mashup

The local search mashup process models how information from an external RESTful API can be processed in order to be visualized on a map widget. The process contains three tasks: 1) retrieve the search results with a GET request on an external resource representing, e.g., the Google Search API; 2) process the results (Geocode) so that they can be converted to a format which is suitable for plotting them on a map; 3) generate an HTML page with the map and the results.

The two versions of the process shown in Figure 7 differ in terms of how they model how the result of the process is made available to its clients. Version (a) makes explicit use of an external resource to store (with a PUT request) the results of the mashup. The lifecycle of the resource is completely decoupled from the one of the process using it to store its results, meaning that the



(a) Result published to external resource (b) Process published as a resource

Fig. 7. Local Search Mashup example

BPMN engine can use its own mechanism to start the process and manage its state. Once the state of the process instance is cleaned up, the external resource carrying its results is still available. Conversely, in Version (b) the mashup process model is published as a resource. Therefore clients can retrieve (`GET /mashup/{instance}`) at any time its execution state, as this is associated with the corresponding process instance resource. Once the execution reaches the final Map task, the state of the process resource will also include the output of this task, which thus can be retrieved by clients (even if the task is not explicitly published as a resource). Once clients DELETE the process instance resource, the output of the mashup will no longer be available.

4.2 Loan Approval

We use the classical loan approval process to illustrate how a business process model can make use of the proposed notation to publish some of its tasks as a resource and to interact with external resources. The process as a whole is published as a resource, which is identified by the `/loan` URI. Two of its tasks (called `choose` and `approve`, marked with the resource symbol in Figure 8) are published as resources. The other tasks are not visible from clients but carry out important back-end activities, such as checking the validity of incoming loan applications, contacting different banks for the latest rates as well as confirming the loan, if an offer has been chosen by the customer and approved by management. Both the retrieval and the confirmation tasks are backed up by an external resource which belongs to the Bank Web Services swimlane.

The notation helps to distinguish that the first interaction (getting the current rate) is a single read-only GET request, while the final confirmation is an idempotent PUT request. The notation could be further refined to indicate that

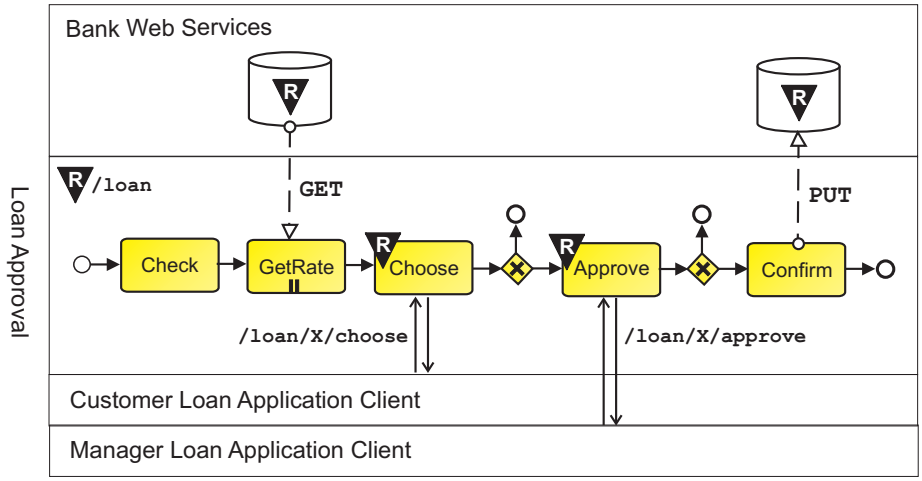


Fig. 8. The Loan approval process example

the confirmation URI was dynamically discovered by the loan approval process, as it could have been provided by the Bank Web Service as a hyperlink found in the response to the GET rate request.

The interaction between the choose and approve tasks with the corresponding clients happens through several request-response rounds as described in the semantics of the "task published as a resource" extension. The pair of edges going from the task to the swimlane representing the client abstract an arbitrary number of requests to the task resource (i.e., GET the current state of the task, or PUT the task in another state) which can happen during the entire lifecycle of the whole process. Such interactions may be allowed or disallowed depending on the current state of the tasks (e.g., once a task has been PUT into a completed state, further state changes will be restricted). This should explain why the shape of the edges is different than the "message flow" edges chosen to represent a single request-response interaction with an external resource.

4.3 RESTBucks

The RESTBucks example is adapted from [19]. It was one of the original case studies advocating the practical usage of REST and hypermedia to guide clients in discovering and following complex distributed workflows. As shown in Figure 9, the BPMN for REST extensions are also suitable to visualize the interaction between a customer and the RESTBucks order management process.

Clients can download a menu (with GET), choose a flavour of coffee (local decision task) and follow a hyperlink to place an order with a POST request to the RESTBucks process, which has been published as a resource. The newly started process instance collects the order request and uses it to compute a price, which is passed to the payment task. The customer can retrieve the price to be paid with a GET request on the payment task, since this has been published as a resource. The response also contains the form to be filled out with the payment details, which can then be submitted with a PUT request also to the same Payment task. Once the PUT request reaches the payment task, its execution completes and the payment information can be validated. If the payment validation is successful, a receipt is produced and stored in the corresponding resource. The client can track the status of the order by GETting the corresponding process instance resource at any time, eventually this status will also contain a link to the receipt resource, which can also be retrieved by the client. Since it is modeled as an external resource, the payment resource will remain available even if the process instance is deleted.

The model of the RESTBucks process also includes the ability to handle updates to the order once it has been created. These can be submitted by clients using a PUT request on the order process instance resource. Such requests will trigger the recalculation of the price by making use of the new resource request event. However, once the payment is received, it is no longer possible to change the order, as modeled with the event sub-process handling the PUT request (which will exit as soon as the Payment task completes).

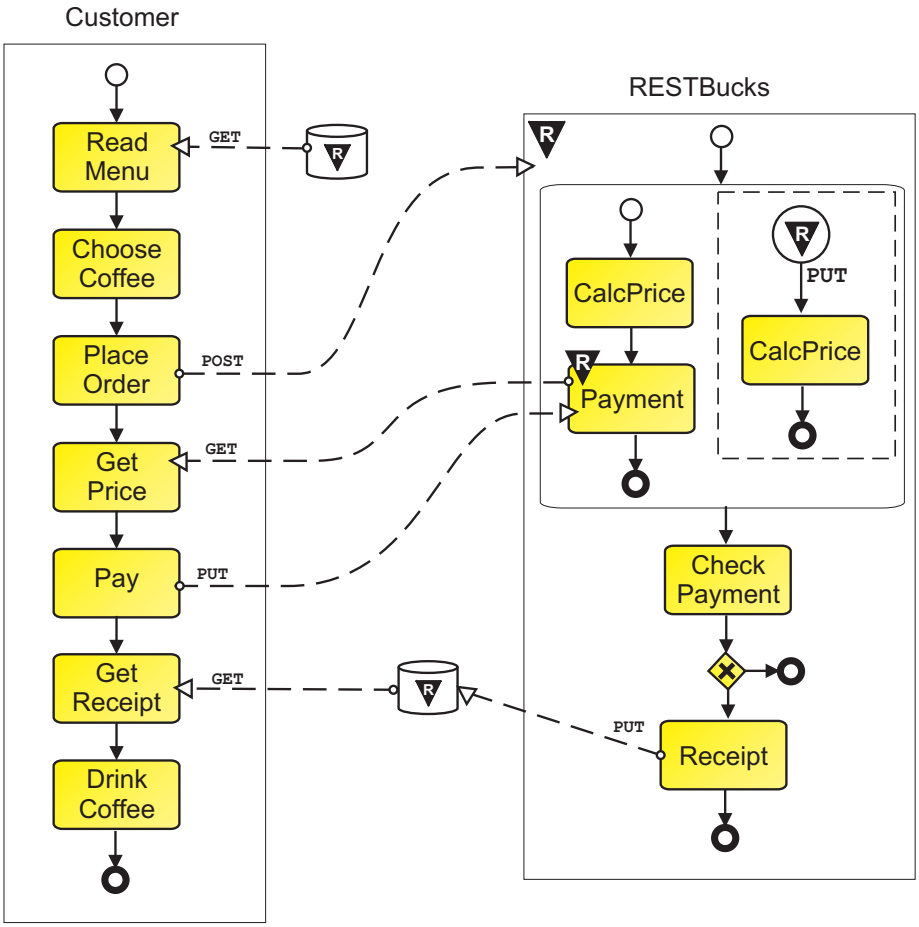


Fig. 9. The RESTBucks order management process example

5 Related Work

This paper shares a similar motivation with our previous work on the BPEL for REST [14] extensions. The concept of using the BPEL language to control the state of resources was first proposed in R-BPEL [12]. The idea of a RESTful Web service API to access the state of workflow instances has been also described in [22], where similar predefined interaction patterns to instantiate new processes where described. A similar idea has been followed in the implementation of the HTTP API of the Activiti BPMN engine.

6 Conclusion

This paper informally sketches the graphical syntax and extended semantics of an proposal for applying the BPMN notation to model RESTful business processes. The goal is to give a precise, expressive yet simple representation of processes which interact with external resources (such as RESTful Web services APIs), and to specify with various degrees of refinement which elements of a process model (tasks, sub-processes or even entire processes) can be published as a resource. Whereas the extensions have a minimal impact on the complexity of the visual syntax of the standard notation, they can already be useful to express several non-trivial RESTful business process model examples.

7 Future Work

Further research is needed to refine the extension to support more dynamic aspects of RESTful business processes, which include features such as: late binding of tasks to dynamically discovered resource identifiers, content-type negotiation and generalized support for hypermedia protocol design. More in detail, we are working on defining the specific meta-model elements associated with the proposed notational elements and have started to look into the problem of how to verify that tasks associated with request events can satisfy the safety and idempotency properties of the corresponding methods. We particularly welcome the feedback from the community concerning the specific advantages or disadvantages of using the proposed notation extension in order to set up a more detailed usability and usefulness analysis.

Acknowledgements. The authors would like to thank the anonymous reviewers for their positive and constructive feedback. This work is partially supported by the S-CUBE network of excellence (EU-FP7-215483) and by the Swiss National Science Foundation with the CLAVOS - Continuous Lifelong Analysis and Verification of Open Services project (Grant Nr. 200020_135051).

References

1. Allamaraju, S.: RESTful Web Services Cookbook. O'Reilly & Associates, Sebastopol (2010)
2. Assmann, U.: Invasive Software Composition. Springer, Heidelberg (2003)
3. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
4. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine, California (2000)
5. Fielding, R.T., Taylor, R.N.: Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002)
6. Foster, I., Parastatidis, S., Watson, P., McKeown, M.: How Do I Model State? Let Me Count the Ways. Communications of the ACM 51(9), 34–41 (2008)

7. Humphrey, M., Wasson, G.S., Jackson, K.R., Boverhof, J., Rodriguez, M., Gawor, J., Bester, J., Lang, S., Foster, I.T., Meder, S., Pickles, S., McKeown, M.: State and events for Web services: a comparison of five WS-resource framework and WS-notification implementations. In: Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), pp. 3–13 (2005)
8. Lessen, T.V., Leymann, F., Mietzner, R., Nitzsche, J., Schleicher, D.: A Management Framework for WS-BPEL. In: Proc. of the Sixth European Conference on Web Services (ECOWS 2008), pp. 187–196 (2008), http://dl.acm.org/citation.cfm?id=1488724.1488774
9. Leymann, F., Roller, D., Schmidt, M.T.: Web services and business process management. IBM Systems Journal 41(2), 198–211 (2002)
10. zur Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 465–479. Springer, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-69534-9_35
11. OMG: BPMN: Business Process Modeling Notation 2.0. Object Management Group (2010)
12. Overdick, H.: Towards Resource-Oriented BPEL. In: Proc. of the 2nd ECOWS Workshop on Emerging Web Services Technology (WEWST 2007) (November 2007)
13. Pasley, J.: How BPEL and SOA Are Changing Web Services Development. IEEE Internet Computing 9(3), 60–67 (2005)
14. Pautasso, C.: RESTful Web Service Composition with BPEL for REST. Data & Knowledge Engineering 68(9), 851–866 (2009)
15. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In: Huai, J., Chen, R., Hon, H.W., Liu, Y., Ma, W.Y., Tomkins, A., Zhang, X. (eds.) 17th International World Wide Web Conference, pp. 805–814. ACM Press, Beijing (2008)
16. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly & Associates, Sebastopol (2007)
17. Vinoski, S.: RPC and REST: Dilemma, Disruption, and Displacement. IEEE Internet Computing 12(5), 92–95 (2008)
18. Vinoski, S.: Serendipitous Reuse. IEEE Internet Computing 12(1), 84–87 (2008)
19. Webber, J., Parastatidis, S., Robinson, I.: REST in Practice: Hypermedia and Systems Architecture. O'Reilly & Associates, Sebastopol (2010)
20. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.: Web Services Platform Architecture. Prentice Hall (March 2005)
21. Wilde, E., Pautasso, C. (eds.): REST: From Research to Practice. Springer, Heidelberg (2011)
22. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing Web Services Choreography Standards — The Case of REST vs. SOAP. Decision Support Systems 40(1), 9–29 (2005)