An Approach for Composing RESTful Linked Services on the Web

Mahdi Bennara, Michael Mrissa, Youssef Amghar

Composing RESTful Linked Services on the Web

Mahdi Bennara
Université de Lyon, CNRS
INSA-Lyon, LIRIS UMR5205,
F-69621, France
mahdi.bennara@liris.cnrs.fr

Michael Mrissa
Université de Lyon, CNRS
Université Lyon 1, LIRIS
UMR5205, F-69622, France
michael.mrissa@liris.cnrs.fr

Youssef Amghar
Université de Lyon, CNRS
INSA-Lyon, LIRIS UMR5205,
F-69621, France
youssef.amghar@liris.cnrs.fr

ABSTRACT

In this paper, we present an approach to compose linked services on the Web based on the principles of linked data and REST. Our contribution is a unified method for discovering both the interaction possibilities a service offers and the available semantic links to other services. Our composition engine is implemented as a generic client that allows exploring a service API and interacting with other services to answer user's goal. We rely on a typical scenario in order to illustrate the benefits of our composition approach. We implemented a prototype to demonstrate the applicability of our proposal, experiment and discuss the results obtained.

Keywords

RESTful Web services; linked services; semantic Web; composition

Categories and Subject Descriptors

H.3.5.2 [**Web-based Services**]: RESTful Web services; Service discovery and interfaces

1. INTRODUCTION

During the last decade, the emergence of Web services has been a major success to enable interoperability on the Web. For almost every simple task we would make on the Web, we can easily find one or several Web services that realize this task. Moreover, service composition, or mashups, enables valued-added processes that combine several services to answer complex user needs. The success of Web services is highlighted via Web sites such as ProgrammableWeb[1] that referenced 105 APIs available on the Web in 2005 and more than 10000 APIs in 2014, not counting mashups.

However, while service-oriented computing becomes adopted, the supporting technologies change. During the last few years, the typical Web service protocol stack (SOAP, WSDL, UDDI) is slowly being abandoned for the profit of REST-based approaches. On the 14/01/2014, ProgrammableWeb counts 2125 SOAP-based APIs for 6833 REST-based ones. A RESTful service respects several constraints [4]:

- It exposes the operations a service offers as a set of one or more Web resources (resource identification).
- It links to others resources to allow for further interaction according to the HATEOAS principle (resource linking).
- It follows the semantics of HTTP verbs to interact with the exposed resource (stateless interaction), thus unifying the interaction possibilities (uniform interface).
- It exchanges self-describing messages.

The adoption of RESTful services changes the way services are designed and implemented. The uniform interface, that comes with the correct use of HTTP verbs and their semantics, replaces the typical API built around functions and input/output parameters. The management of the application state, which was usually handled server-side, is now at the charge of the client software (the browser).

In addition, recent advances in the semantic Web research area have been promoting linked data [3] and a set of languages and tools such as JSON-LD [13], RDF [10], OWL [5], SPARQL [15] and POWDER [2], that allow to annotate Web data, resources and services with explicit, machine-readable semantics that can be utilized in conjunction with advanced reasoning mechanisms to connect resources to each other in a way that makes sense. Indeed, Web services also benefit from these advances, and can now be annotated with machine-readable, explicit semantics and exchange semantically annotated data. These services are referred to as linked services [12][2]. Therefore, the typical approaches to discover, compose, orchestrate and utilize services on the Web need a complete rethinking to comply with -and get the benefits of- the REST principles and the semantic Web. Reaping the benefits from these advances requires adapting the way information systems and applications interact with the Web.

In this paper, we propose a solution to facilitate service composition based on the REST and linked data principles. In the remainder of this paper, services are semantic Web resources, they are identified via a URI and they are accessed through HTTP verbs. They also are semantically described and represented as JSON-LD documents to make semantics explicit when needed. The contribution of this paper is to propose a unified approach to discover the interaction patterns a resource offers and the orchestration possibilities with other resources, in order to enable resource composition, using the principles of REST and linked data.

Our paper is organized as follows. Section 2 introduces the scenario to illustrate our proposal and gives details on the motivation of our contribution. Section 3 details our contribution and shows its innovation. Section 4 shows how our prototype operates in the context of our scenario and demonstrates the applicability of

[1] `http://www.programmableweb.com`

[2] Linked services are Web services that exchange linked data and are described with linked data. We do not use the term linked data services to avoid confusion with data services that only provide data.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
`http://dx.doi.org/10.1145/2567948.2579222` .

our solution. Section 5 presents related work and highlights the advantages our solution offers. Section 6 discusses our results and gives some guidelines for future work.

2. SCENARIO AND MOTIVATION

In order to motivate our approach, we consider a typical scenario similar to the scenario presented in [11] involving a customer who buys a book from an online seller. At the beginning of the scenario, we assume the user only knows the URI of the book selling service. The user also knows his/her objective: ordering a book, paying online and getting delivered in less than 48h if possible. After choosing a book and creating an order, the client must choose a specific shipment method, give the address to which he/she wants to be delivered, and finally perform the payment operation.

In this scenario we identify a set of elementary services, which are, according to the REST principles, exposed as resources. The resources modeled within this scenario are the Book, Order, Shipment Method, Shipment and Payment resources.

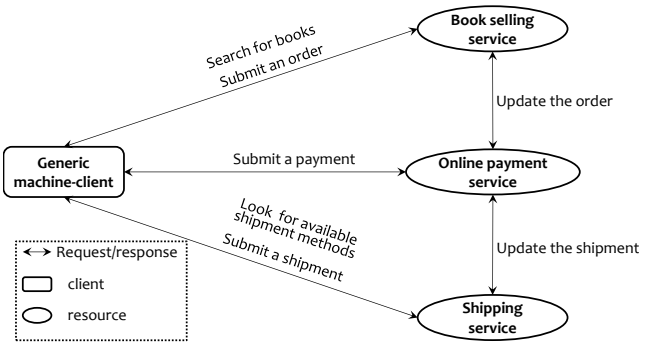

Figure 1: Resources and data exchanges in our scenario

Fig. 1 shows the resources and the different exchanges involved in the scenario, described as follows:

- Book selling service: it is composed of two sub-resources, a list of books and a list of orders. Book descriptions can be retrieved using GET and new orders can be created using POST and updated with the list of books to buy and payment confirmation with PUT (or PATCH)
- Payment service: a new payment can be created with POST and confirmed with PUT (or PATCH)
- Shipment service: a shipment method can be selected with GET and a new shipment can be created with POST and validated with PUT (or PATCH)

In a typical service-oriented architecture, the user's high level objective is described in a business process that is located server-side and orchestrated/executed by the composition engine. Service orchestration is typically either hard-coded under the form of a business process (or mashup) or dynamically generated on the server side using well-known reasoning techniques. Such a solution is not an option according to the REST principles as the handling of the composition needs to be performed client-side, relieving the server from handling the composition process and offering better scalability. Therefore, this scenario underlines the following scientific locks that motivate our work:

- There is a need for the user to interact with all these services using the same generic client.
- There is also a need for the client to be able to use several services together without manually connecting them, and to locally handle the state of the composite application.
- There is a need to provide semantically explicit links that allow the generic client to go from one service to another.

This scenario highlights our motivation: we want to show how the service discovery and composition tasks can be achieved using the Web, linked services and the REST paradigm. To do so, we provide a unified method for discovering resource interaction possibilities as well as external semantically linked resources that can be used in conjunction. This contribution is a first step towards automated discovery and composition of linked resources (or RESTful linked services) on the Web.

3. CONTRIBUTION

In this section, we explain how our unified method explores the features resources offer in terms of what HTTP operations are allowed, and how it allows discovering other resources that are semantically linked to the resource currently explored, according to the composition possibilities they offer. Then, it becomes very simple for a client-side software program to interact with resources and crawl from a resource to another to fulfill the user's objectives.

3.1 Resource Interaction Model

Our model involves specific interaction possibilities with resources on the Web. We extend a resource with a descriptor that contains meta-data about the resource together with information about related resources. Then, when the URI of a resource is available, it is possible to get its descriptor too, following a generic interaction pattern. To make this interaction possible, we use the HTTP LINK header and the POWDER [2] `describedby` property, expressed in RDF[3], in order to give a link with explicit semantics to the descriptor of a resource. Fig 2 shows the discovery of a resource and how the LINK field is accessed in the HTTP header.

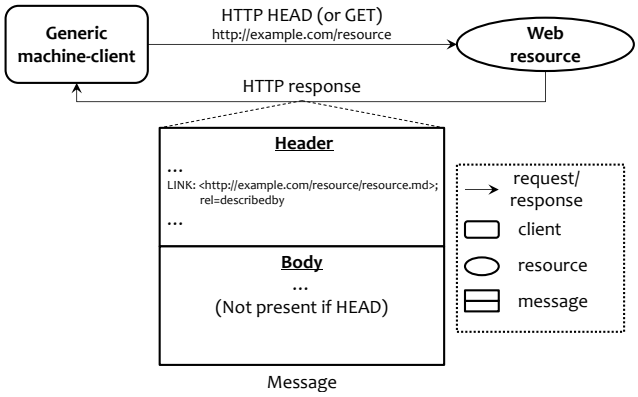

Figure 2: Discovering a resource

The descriptor describes the resources pointing at it by giving information on what HTTP operation are available on the resource, together with information on available sub-resources and on other related external resources, as illustrated in Fig. 3.

We must underline the fact that all descriptors are resources also according to the REST principles. Then, as the descriptor is also

[3]Alternative solutions are available at `http://www.w3.org/TR/powder-dr/#httplink`

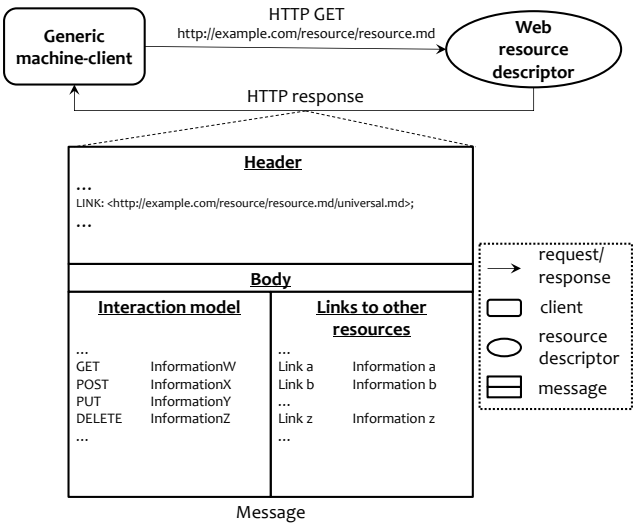

Figure 3: Discovery of a resource descriptor

modeled as a resource, the mechanism recursively applies. According to our proposition every resource must have a descriptor containing meta-data about it. One can get the descriptor of a descriptor via the Link field of the HTTP header received after a GET or HEAD operation. However, in such a case, the body of the answer contains meta-data about the descriptor, and the Link field in the HTTP header contains a link to the universal descriptor that describes all descriptors[4]. Figure 4 shows how resources, descriptors and the universal descriptor are linked to each other. At the moment, the meta data that describes a descriptor does not provide link to external resources, but such a possibility could be explored, for example for meta-data crawling.

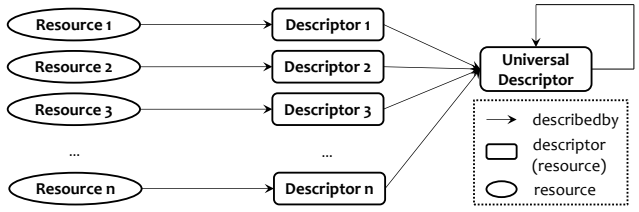

Figure 4: Links between resources, descriptors and the universal descriptor

3.2 Unified Resource Discovery

The way clients get access to internal and external resources is homogeneous. A client accesses related resources with the required information about each of them, helping to decide which path to follow.

The information available in the link to another resource must be helpful for the client to decide whether it should interact with the target resource or not. It must describe the semantics of the relation with the current resource and any other useful piece of information. However, the link information must not give too much details on the resource. Details about a resource can be found in its own descriptor.

[4]Note that a GET or HEAD on the descriptor of all descriptors returns a link to itself. As the universal descriptor is at the same time a descriptor, it should imperatively describe itself.

In addition, giving too much information at this stage can generate data redundancy or bandwidth related issues. The best compromise is to only give the information that allows the client, at a given point, to decide whether to use the linked resource or not.

As well, describing the HTTP operations allowed on a given resource with information on how to correctly use the corresponding operation is useful. Our work relies on the Siren project [14] in order to describe resources, links and HTTP operations. The information consists first of all in a description of the data-models expected and to be returned when the HTTP operation is called. The HTTP status codes that can be returned with additional details on the circumstances of each code gives details on the use of the HTTP operation. Other annotations on the operation itself, such as the CRUD basic operations (creates/reads/updates/deletes) and other advanced operations (initializes, cancels, confirms, etc.) can be useful for certain clients. Annotations on the data manipulated by the operation makes its semantics explicit in order to allow a better exploitation by the client. The HTTP operations that can be executed on the current resource can also be detailed in order to allow clients for unambiguous interaction. In particular, the semantics of the POST operation can be semantically linked to explicit descriptions to help the client drive the interaction.

While the definition of precise semantics is to be completed in future work, we envision also to semantically annotate the relation to external services with several properties that characterize how the other resource can be used with the original resource. We deem appropriate to use different interaction patterns such as precedes, follows, complements, replaces, and so on, that may help the user know what is the next link to follow and discover. These different patterns help clients build the different paths available to our generic client during a composition Additional information on the resource itself, such as provider, description, service area, language, location etc. can be integrated to the descriptor as well, but such details remains out of the scope of the current paper. Our resource discovery and composition model relies on the fact that resources can keep track of the interactions they have been involved in, and publish these traces, after processing, via their descriptor. Several possibilities exist to record resource interaction, such as storing the addresses of incoming requests, these addresses may lead to other related resources. Another solution is to extract well-known mashups from Web sites such as ProgrammableWeb. In this paper, we assume the list of connectible resources is maintained for each resource available on the Web. How to build and maintain such a list is subject to future work.

The applicability of our proposal and in particular of the unified discovery process is demonstrated with our scenario. We implement a generic client that takes advantage of the given descriptors in order to decide, at every moment and according to its user's needs, what resource links it should follow. While interacting with the different resources, the generic client handles the application state until it fulfills the user's goal. Figure 5 shows how our generic client discovers the different resources needed in our scenario.

4. IMPLEMENTATION

In the following, we first present our implementation setup, then we explain how the scenario runs with this setup. We give details about how each service is implemented and how it interacts with the client software and the other services using the elements of our contribution. Our prototype implementation is available on our development platform [5].

[5] `http://soc.univ-lyon1.fr`

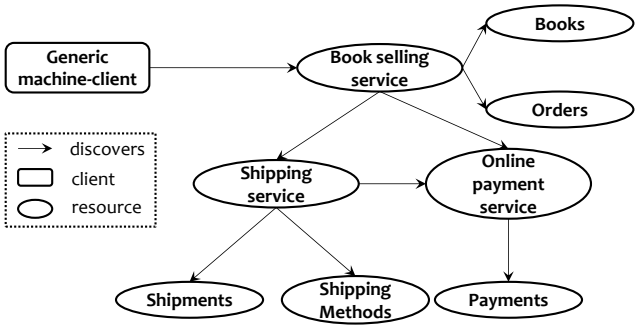

Figure 5: Service Discovery in our Scenario

4.1 Setup

We implemented the services defined in the scenario using the Java™Jersey framework for developing RESTful Web services based on the JAX-RS API. We used Eclipse as an IDE and the Glassfish application server to accommodate our services. We have developed 3 services, each in a single Java™project, to keep the services independent from each other. Other service URIs are obtained by retrieving the book selling (the entry point) descriptor that leads also to its own resources, books and orders. The communication between the generic client and the services is ensured using the `application/ld+json` media type. We use the json/POJO serialization in order to convert the data transferred into a manipulable format in Java. We currently rely on the Siren project[14] to describe Web resources.

Our client is currently a Java application but we plan to quickly implement it in Javascript as a browser plugin. The client takes the URI of the first resource to discover as input. According to our scenario and contribution, the client follows links found in the meta-data retrieved from the entry point (book selling service resource) and calls HTTP operations on the right resources with the right data in order to answer the client needs. At the moment, this process is manually performed via user interaction through the client interface, but future work includes developing a reasoner to be able to reason about the user's objectives and match these objectives with the states of the different resources available to the client.

The structure of the descriptor is implemented via a Java class that encapsulates both allowed HTTP operations of the resource with its associated information and links to other external and internal resources also with their associated information, as well as the properties of the resource itself. Therefore, all the descriptors as well as the universal descriptor are instances of this class. They are considered as resources just like the three services of the scenario.

4.2 Revised Scenario

We present here the scenario and how it operates according to the elements of our contribution. Typically, a HEAD (or GET) HTTP operation on any resource should return the URI of the resource description (`http://example.com/resource/resource.md`, where md stands for meta-data) in the LINK field of the HTTP header, as described in Fig. 2. A GET operation on the URI of the link returns the descriptor, which is also a resource. The descriptor contains links and all the necessary information about the sub-resources of the resource it describes, and about external related resources. As we talk about a unified method for semi-automatic discovery, the descriptions of the internal sub-resources and external ones are homogeneous and processed the same way on the client side. The description allows the generic client to explore both internal and external interaction possibilities according to the information given on each resource. The client knows what is the best HTTP operation to apply to the resource according to its needs and the information available in the obtained description. A discussion on resource description is provided in section 5.

In the following, we present the resources modeled in our scenario and give their corresponding URIs. The particular resources are stored in their respective repositories (books, shipping methods, shipments, payments) and have the following URI patterns:

- Particular book: /bookselling/books/book123
- Particular orders: /bookselling/orders/order123
- Particular shipping methods: /shipping/methods/method123
- Particular shipments: /shipping/shipments/shipment123
- Particular payments: /payment/payments/payment123

In our scenario, the user first wants to browse book descriptions[6] (GET on `books repository` resource) to select one or several `book` resources, whose detailed descriptions are then stored client-side. The problem is that the client at first does not know the URI of the books repository. It discovers this URI using the descriptor of the book selling resource.

The link to this descriptor is found in the HTTP response to a HEAD (or GET) operation on the entry point, we suppose known by the client, represented by the URI of the book selling service /bookselling/. More precisely, in the very beginning of the execution flow, the client executes a HEAD operation on the entry point URI. It gets the URI that corresponds to the `describedby` property attached to the Link Header field of the HTTP response, which is /bookselling/bookselling.md in our scenario. Then, it performs a GET operation on this link to get the descriptor of the book selling service. In this descriptor, the client finds multiple links to other related resources. It chooses the one pointing at the book repository using the semantics provided with the link and according to the user's needs.

Another question that our contribution answers to is: how does the client know what HTTP operation to execute in order to obtain book descriptions. The answer is: the client consults the meta-data on allowed HTTP operations and decides which one responds to the objective of getting a book description, in this case it is the GET operation.

Second, there is a need to create an `order` (POST) on the `orders repository` resource. The latter is referenced via the meta-data contained in "bookseller.md" retrieved from the book selling also exactly the same as the `books repository` resource. The HTTP operation to execute is retrieved exactly as explained above.

Once the `order` resource created, it remains in the status "Unpaid". It can be updated (PUT) to "Paid" once the payment is confirmed[7]. Then, the link to the shipment resource is extracted with the help of the "bookseller.md" resource that includes external resources.

Third, a shipment method is selected (GET) and a new `shipment` is created (POST) and set to the status "Unpaid", waiting for an update on the payment confirmation (PUT). The link to the "payment" resource is also retrieved by reading the "bookseller.md" resource.

[6]We refer here about the actual book description with its URI, author, title, price, and so on.

[7]For the sake of brevity, we omit security concerns, but such an exchange may involve key sharing and secured protocols such as Oauth (`http://oauth.net/`).

Finally, the `payment` resource is created (POST) and confirmed as effective (after getting a response from the `order` and `shipment` after the PUT operation).

5. RELATED WORK

5.1 Invocation of RESTful Services

The Hydra project [9] proposes a generic client to discover and invoke the different HTTP operations Web resources offer. The client relies on the Hydra core vocabulary and on the JSON-LD serialization format for data exchange.

One of the main advantages of Hydra is the possibility to build a generic client. Hence, there is no need to develop and compile a specific client to run through the Web API of each service. Also, Hydra proposes a mechanism that allows Web APIs developers to document their Web APIs into a format that can be either interpreted by machine or read by humans. The machine also can generate a human-readable format based on the documentation. The main drawback of Hydra is that it requires an effort in order to adapt existing solutions, and that the model developed is specific to Hydra.

The Siren project [14] is another effort aiming for representing entities as Web APIs using the JSON format. An entity models a URI-addressable resource. The description of an entity includes a `class` field that describes the concept of the entity (for example `book`), a `properties` field that contains a set of key-value pairs describing the entity , an `entities` field that contains a list of related sub-entities, a `links` field that lists items containing navigational links including link to itself, an `action` field that lists a set of available actions on the current entity, described with the corresponding HTTP method, and a `title` field that textually describes the entity in question. Siren descriptions answers our needs for the description of Web resources and have been reused in our work.

The principle of distributed affordance [16] allows user interaction with resources in a distributed way. An affordance is defined as the possibility to perform an action over a resource. The authors define an architecture that enables enriching resource representations with distributed affordances. The affordance creation process happens based on information about a given resource, for example a book title, and knowledge acquired about user preferences and action providers, for example a local library service. The architecture is provided as a service, based on an API description catalog to gather action possibilities as well as user preferences. This solution is different from our approach where such information is directly attached to resources via their descriptors.

RESTdesc [17] is another effort for semantic description of Web APIs. The purpose of the description is to allow for an efficient way to discover the different features Web APIs offer. The RESTdesc approach takes into account the principles of REST. Since REST is based on the correct use of the HTTP protocol, RESTdesc format relies on the HTTP vocabulary in RDF [8] to describe the semantics of HTTP exchanges. The main advantage of RESTdesc is the loose coupling and reuse of existing models it offers.

The main benefit of our approach is to require only a few efforts in order to apply to existing Web services. The work to carry out is to add to the header fields of the HEAD or GET HTTP response of every resource a Link element that contains the URI of the resource descriptor. Building the descriptor contents also does not require a lot of work. In fact, the properties of a resource itself are already well known. For the HTTP operations allowed, as we run through the service source code we can easily show up the available HTTP operations as well as the properties that go with it. Concerning the list of links to other resources, there are a multitude of approaches to build such a list, as mentioned in Section 3.

5.2 Description of RESTful Services

ReLL [1] is a solution for the description of RESTful Web services that focuses on their Hypermedia property (HATEOAS). For ReLL, a RESTful service is a set of resources related to each other. Every resource in the set has its unique identifier in addition to its name, description in human readable language and other optional properties. A resource may have different representations depending on the media-type or syntax in use and every representation may contain links to other resource representations. The application domain semantics are explicitly supported in a resource representation. These semantics are made explicit using resource annotations.

LRDD [6] is an effort to describe and obtain information on Web resources that are identified by URIs. The resource descriptor link is indicated by the resource itself using different methods such as with the Link field of the HTTP header. The link to the descriptor is dependent on the resource URI. The content of the descriptor is a machine readable information allowing better interoperability and enhancing the interactions with the resources.

HTTP vocabulary in RDF [8] is a W3C working draft aiming to represent the HTTP protocol in RDF format. It aims to link the REST paradigm with the concepts of the semantic Web. It gives a set of RDF classes and properties aiming to represent the HTTP specification as concepts.

The descriptions we use in our work are mainly inspired from the work cited above. We aim to reuse existing formats as much as possible in order to enhance the interoperability of our solution.

5.3 Discovery of RESTful Services

The RESTdesc [17] discovery process is based on the HTTP OPTIONS verb. The invocation of this operation on a given resource returns back information in Notation3 Syntax. This information informs the machine-client of the current resource interaction possibilities. Since OPTIONS presents some drawbacks [8], an alternative is to link to descriptions using the LINK field of the HTTP header.

The discovery mechanisms in RESTdoc [7] format distinguishes two different aspects of the RESTful service discovery. Discovery as a client, or discovery as you browse, concerns client-side browsers. This kind of discovery relies on HTML Link elements on a Web site in order to point to other resource descriptions. Discovery as a service, also called automated discovery, is the ability for a service to access and link to other related resources in the same application domain. The main idea is to construct a graph by running through links and identifying resources. This graph can be subject to later extension to explore related resources. The RESTdoc solution relies on a microformat-like syntax to annotate resource representation directly in the HTML source and on an adapter to convert into RDF.

In our solution, the discovery mechanism helps find the path that clients may follow, through the progressive exploration of resource descriptors accessible from the HTTP Link Header embedded in HTTP responses. The resource meta-description is clearly separated from the resource representation and located in the descriptor. Resource discovery is performed locally from a given resource by consulting the list of related resources. The generic client guides the discovery process as it gets the list of related resources and decides whether or not to follow a given link according to the user's needs.

5.4 Composition of RESTful Services

Several works have been carried out in order to resolve the composition problem for RESTful Web services. Some of these works propose to reuse the BPEL language as mentioned in [11].

[8]As mentioned in `http://www.mnot.net/blog/2012/10/29/NO_OPTIONS`

There are two different ways to use the BPEL principles to compose RESTful Web services. The first one suggests to use the WSDL 2.0 description language without changing the current BPEL. The solution relies on the new HTTP binding element introduced in WSDL 2.0. In fact, the RESTful Web resource API is wrapped behind a WSDL document that acts as an interface between the REST Web resource and the BPEL code, using the HTTP binding with the REST resource and operation invocation on the BPEL side. The second way to use BPEL with REST resources is more direct than the first one but it requires an extension to BPEL in order to be able to support the HTTP operations on the resource API. However the main drawback of BPEL comes with its centralized approach. In fact, it relies on a static composition engine, which does not fit with the REST principles that rely on the resource notion and HATEOAS. Additionally, a centralized execution process makes it less interesting in the large scale context of the Web, for which RESTful Web services have been designed.

Another approach is proposed in [18] for composing RESTful Web services that takes full advantage of their characteristics. The authors propose a situation calculus-based state transition system that ensures the automatic composition for RESTful Web services. The basic idea is to take the composition problem and transform it into as state transition problem. The latter can be resolved using the situation calculus principles and components. However this proposal does not mention any implementation, hence it is difficult to evaluate the solution in terms of complexity, accuracy and response time, as well this kind of approach is centralized and presents the same drawbacks as the BPEL-based approach.

6. CONCLUSION

In this paper, we have proposed a novel approach to compose RESTful linked Web services. Our proposal relies on the description of Web resources in order to allow a unified discovery process for both internal sub-resources and external services. Our generic client is responsible for carrying the composition steps by getting resource descriptions, discovering new resources and sending data from one resource to another. The composition process is guided by the generic client according to the user's needs. Our scenario motivates our contribution and demonstrates the feasibility of our approach. Our proposal consists in attaching a descriptor resource to each Web resource. A link to the descriptor is obtained from the LINK header of a HTTP HEAD (or GET) response from the resource URI. A descriptor contains useful information about the resource and how it interacts with clients and other resources. It contains links and information about internal and external resources. By following these links and making use of the available information, the client handles the composition process in order to fulfill the user's objectives. As future work, we aim to focus on how to use reasoning mechanisms to process a high level client request into a set of composition steps. The user request should be processed by a reasoner, and the output of this analysis should be a set of partially ordered HTTP requests to perform on resources, in order to answer to the initial user request. Also, as future work, we plan to extend the resource description with additional information on related resources such as their availability, quality, average response time in order to facilitate resource selection.

7. REFERENCES

[1] Alarcón, R., Wilde, E., Bellido, J.: Hypermedia-Driven RESTful Service Composition. In: Maximilien, E.M., Rossi, G., Yuan, S.T., Ludwig, H., Fantinato, M. (eds.) ICSOC Workshops. Lecture Notes in Computer Science, vol. 6568, pp. 111–120 (2010)

[2] Archer, P., Smith, K., Perego, A.: Protocol for Web Description Resources (POWDER): Description Resources. W3C Recommendation, http://www.w3.org/TR/powder-dr/

[3] Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)

[4] Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000), aAI9980887

[5] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: Owl 2 web ontology language primer. W3C Recommendation, http://www.w3.org/TR/owl-primer/

[6] Internet Engineering Task Force (IETF): LRDD Internet Draft. https://tools.ietf.org/html/draft-hammer-discovery-06

[7] John, D., Rajasree, M.S.: RESTDoc: Describe, Discover and Compose RESTful Semantic Web Services using Annotated Documentations. International Journal of Web & Semantic Technology (IJWesT) 4(1) (2013)

[8] Koch, J., Velasco, C.A., Ackermann, P.: HTTP Vocabulary in RDF 1.0. W3C Working Draft,, http://www.w3.org/TR/HTTP-in-RDF10/

[9] Lanthaler, M., Guetl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., Auer, S. (eds.) LDOW. CEUR Workshop Proceedings, vol. 996. CEUR-WS.org (2013)

[10] Manola, F., Miller, E.: RDF Primer. W3C Recommendation, http://www.w3.org/TR/rdf-primer/

[11] Pautasso, C.: RESTful Web service composition with BPEL for REST. Data Knowl. Eng. 68(9), 851–866 (2009)

[12] Pedrinaci, C., Domingue, J.: Toward the Next Wave of Services: Linked Services for the Web of Data. J. UCS 16(13), 1694–1719 (2010)

[13] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Lindström, N.: JSON-LD 1.0 - A JSON-based Serialization for Linked Data. W3C Recommendation, http://www.w3.org/TR/json-ld/

[14] Swiber, K.: Siren: a hypermedia specification for representing entities. https://github.com/kevinswiber/siren

[15] The W3C SPARQL Working Group: SPARQL 1.1 Overview. W3C Recommendation, http://www.w3.org/TR/sparql11-overview/

[16] Verborgh, R., Hausenblas, M., Steiner, T., Mannens, E., de Walle, R.V.: Distributed affordance: an open-world assumption for hypermedia. In: Carr, L., Laender, A.H.F., Lóscio, B.F., King, I., Fontoura, M., Vrandecic, D., Aroyo, L., de Oliveira, J.P.M., Lima, F., Wilde, E. (eds.) WWW (Companion Volume). pp. 1399–1406. International World Wide Web Conferences Steering Committee / ACM (2013)

[17] Verborgh, R., Steiner, T., Deursen, D.V., Roo, J.D., de Walle, R.V., Vallés, J.G.: Description and Interaction of RESTful Services for Automatic Discovery and Execution. In: Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services (Dec 2011)

[18] Zhao, H., Doshi, P.: Towards automated restful web service composition. In: ICWS. pp. 189–196. IEEE (2009)