Comparing Major Web Service Paradigms

Steffen Stadtmüller, Sebastian Speiser, Martin Junghans, and Andreas Harth

AIFB/KSRI, Karlsruhe Institute of Technology, Germany
`firstname.lastname@kit.edu`

Abstract. We identify three foundational tasks for clients interacting with Web services, and study how to realise these tasks using major paradigms (WS-*, Semantic Web Services, RESTful APIs and Linked APIs). We identify relative merits of each approach and point out how the combination of resource-oriented services with Linked Data can enable an interoperable ecosystem of Web services.

1 Introduction

Services are an integral part of the Web, giving access to functionalities and data in a wide range of scenarios. For example, forms in Web pages give end users access to the "Deep Web", JSON-based APIs enable mashups to combine several Web 2.0 applications and service-oriented computing technology enable businesses to integrate each other's systems.

There exist several approaches that attempt to bring services to the Semantic Web. We survey these approaches and discuss their relative merits. We structure the comparison according to two major classes of Web services as identified in a W3C Note[1]:

- Services that expose an arbitrary set of operations as often designed when adhering to the variety of specifications and languages commonly referred to as WS-* stack.
- Resource-oriented services with a uniform set of stateless operations, that comply to the constraints of a Representational State Transfer (REST) architecture.

We thus classify Web service paradigms firstly according to the distinction between services that allow for arbitrary operations, and services that require a uniform set of operations, and secondly according to syntactic approaches and semantic ones (i.e., those that include logic-based knowledge representation formalisms). Table 1 lists the paradigms we cover.

To compare the approaches, we study the following three basic tasks that service consumers face in their interaction with services: (i) *invocation* which can be seen as the most basic ability a service has to provide to be useful; (ii) the *connection* or the sequential calling of services which is a foundational workflow pattern [1]; (iii) *substitution* of one service for another which reflects the ability to establish uniform interfaces, thus facilitating an ecosystem of fungible services on the Web. To illustrate and compare the different paradigms, we use a scenario where a client first invokes a picture hosting service to upload a picture, then invokes a micro blog service to post information about the picture, and finally replaces the micro blog service with another equivalent service.

[1] `http://www.w3.org/TR/ws-arch/#relwwwrest`

Table 1: Matrix of Web service approaches.

	arbitrary operations	**uniform operations**
semantic	Semantic Web Services	Linked APIs
syntactic	WS-* Services	REST

Section 2 explains how to realise the foundational tasks with services that allow for the definition of arbitrary operations: the WS-* stack and semantic Web service technologies. Section 3 shows how to realise the tasks with services that prescribe a set of uniform operations: RESTful services and *Linked APIs*. Section 4 concludes with a detailed comparison of the different paradigms.

2 Services with arbitrary operations

2.1 WS-*

Approaches that allow for the definition and exposition of arbitrary operations provide an interface for calling methods with corresponding parameters and return values. A prominent specimen is the WS-* stack of standards, including technologies such as SOAP (Simple Object Access Protocol) and WSDL (Web Service Definition Language).

Communication partners in the WS-* world exchange messages using SOAP. While SOAP supports several transportation channels, HTTP is the most widely used. Another standard, WSDL, specifies in an XML document the operations that a Web service provides, including their parameters and return values. The description also contains information about the binding to so-called service endpoints, which allows for the automatic construction of code that provides an interface to the operations.

WSDL documents provide means for syntactic interoperability only, as data modelling is typically performed using XML Schema. XML Schema is a language for defining hierarchical document models, especially their structure and syntax. Thus, WSDL enforces a hierarchical representation on the data which may be seen as less natural than a network-structured modelling approach [2].

2.2 Semantic Web Services

Several approaches extend the existing WS-* stack with semantic capabilities by leveraging ontologies and rule-based descriptions (e.g., [15, 6, 5]) to achieve an increased degree of automation in high level tasks, such as service discovery, composition and mediation. Those approaches extending WS-* became known as semantic Web services (SWS).

One common feature of different SWS approaches is the *lifting* and *lowering* of messages (usually specified in XSLT scripts), which establish the relation of data elements in WSDL documents to terms defined in formal ontologies. Furthermore, SWS approaches model the functionality of services by giving semantic descriptions of inputs, outputs, preconditions and effects.

2.3 Example

In the following, we describe how the proposed scenario can be realised with WS-* services and SWS in general.

For the *invocation* of the picture host service, the developer uses a tool to generate from the corresponding WSDL document a client stub, i.e., code that provides local methods in the target programming language. The tool derives the signatures of the client methods by a mapping from the XML datatypes to types in the target programming language. The developer can program the Web service client as if it would use a local library. Typical problems at this stage include the interpretation of parameters and return values, understanding the functionality of the operations, and error handling. In case of a SWS, we assume that we have a semantically described WSDL-based Web service and that the client uses a semantically described data model internally. The lowering of the input data and lifting the expected returned information needs to be implemented manually, e.g., by an XLST script. Once this mapping is provided, the underlying Web service can be invoked as described for any WSDL service.

To establish the *connection* of the picture service with a micro-blogging service, the developer generates a new client stub from the WSDL description of the micro-blogging service. He extends the local code to first post the image to the picture service, and then create an entry in the micro blog timeline, including a link to the picture, which is returned by the picture service. Typical problems at this stage are the conversion of return values and input parameters of the different services, in this case the URI of the uploaded picture. With SWS, the pattern requires the specification of lifting and lowering schemes for both services. Then, messages returned from the picture hosting service can be processed on a semantic level (which eases the parameter conversion as the client's data model is defined on the same level). Then, the lowered representation of the parameter is passed to the subsequent micro-blogging service. Still, checking the compatibility between subsequent services can take place on the semantic level. Alternatively, one can use the Business Process Execution Language (BPEL) to compose the services following a *programming in the large* approach. BPEL is an XML-based language for process specification, which besides other more complex constructs also supports the sequence pattern, which can be used to connect the picture hosting and the micro-blogging services. Extraction of the right return values of the response of the picture hosting service, can be selected via XPath and transformed via XSLT to generate the correct input of the micro-blogging service. The BPEL process can be deployed on and executed by a BPEL process engine, and exposes the composed functionality again as a WSDL service.

Finally we analyse the *substitution* of the micro-blogging service. Without using BPEL, the developer of a service composition generates a new client stub for the new micro-blogging service. To further use the code of the service composition, the developer has to emulate the methods and data model of the original service, as the automatic tooling used for WSDL-based services leads to a tight coupling of the client to a service. Such a coupling can bring problems when substituting the service, due to differently grained interfaces and incompatible protocols. With SWS, again appropriate lifting and lowering scripts need to be adapted for the replacement service. The generation of a client stub for the integration of the service call into the client's application is required as described for WS-* services.

3 Services with uniform operations

3.1 RESTful APIs

Repelled by the complexity of the WS-* stack, there is a community proposing to move toward a service model that is centered around resources for which the services provide a set of uniform operations. A major role plays Representational State Transfer (REST [7]) as software architecture for distributed systems. In a REST architecture client and server communicate in a synchronous request-response pattern. On the Web, the constrained set of operations consists of *HTTP verbs* (e.g., GET, PUT, POST and DELETE), in contrast to a SOAP communication design, where arbitrary operations can be implemented all using the same verb (POST). A client applies the HTTP operations to resources addressed via individual URIs. A resource is anything that clients can interact with while progressing toward their goal.

Client and server communicate representations of the state of resources. The format of a representation (i.e., content type) is supposed to be negotiated between client and server, which use content type processors to read and produce the resource representations in the supported formats. If client and server agree upon a specific content type, they promise to understand and deliver the corresponding format of the resource representation, thus forming a service contract.

The communication itself is stateless, which means a server immediately forgets about a client after a request is served and a client can not assume that the resources of the response remain unchanged on server after invocation. Every HTTP operation can be performed independently from previous operations, due to the stateless communication.

Another feature of REST, the so-called hypermedia constraint, requires a server to include links in the resource representations, so a client can discover new resources to interact with. The provided links to other resources reflect possible next steps that a client can take.

3.2 Linked APIs

Many approaches to semantically enable RESTful services [8, 14, 16] leverage the advantages coming from the combination of Linked Data technologies[2] and service technologies [10]. These approaches (referred to as Linked APIs) view services primarily as RDF prosumers, i.e., the state of resources is represented in RDF. RDF-based communication enables service descriptions using graph patterns either in N3 or SPARQL. An input graph pattern precisely describes the expected input graph, which is submitted in a request. An output graph pattern formalises the content of the state representation

[2] `http://www.linkeddata.org/`

in the response. The output pattern reuses variables from input pattern to make the relation between input and output explicit. The variables in the output, that do not originate from the input pattern are bound by the service functionality.

3.3 Example

For discussing the implementation of the example using REST technologies we assume the following resources: the uploaded picture (`http://rest-PicHost.org/bucket/pic01`), the timeline of a micro blog (`http://rest-MicroBlog.org/timeline`), and the post on the timeline that informs about the newly uploaded picture (`http://rest-MicroBlog.org/timeline/post45`).

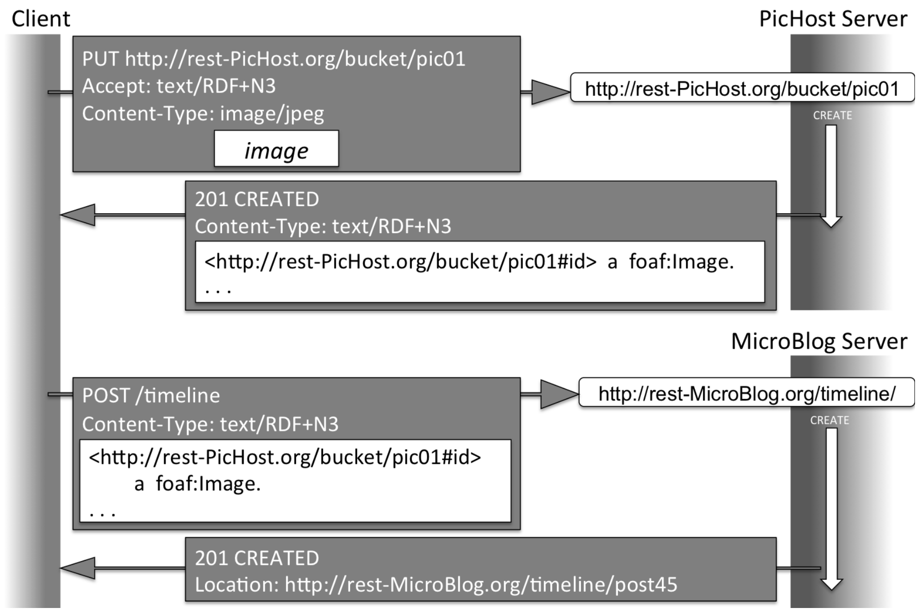
Client
PicHost Server
PUT http://rest-PicHost.org/bucket/pic01
Accept: text/RDF+N3
Content-Type: image/jpeg
image
http://rest-PicHost.org/bucket/pic01
CREATE
201 CREATED
Content-Type: text/RDF+N3
<http://rest-PicHost.org/bucket/pic01#id> a foaf:Image.
. . .
MicroBlog Server
POST /timeline
Content-Type: text/RDF+N3
<http://rest-PicHost.org/bucket/pic01#id> a foaf:Image.
. . .
http://rest-MicroBlog.org/timeline/
CREATE
201 CREATED
Location: http://rest-MicroBlog.org/timeline/post45

Fig. 1: Client-server interaction with a RESTful API.

A client *invokes* a single service via submitting an HTTP request to a URI-identified resource. A client interacting with the picture host API can upload a new picture by performing an HTTP PUT, thus creating a new resource on the server. The request contains the applied HTTP verb and if necessary a body, which holds input data. The request header provides information about the content-type of the input data (e.g., *image/jpeg*) and the content type of the expected output data. The response contains in its header the status code, which provides information about the success of the request. The body of the response contains optionally output data about the resource state (e.g., size of the picture). The difference between the invocation of a traditional REST service and a Linked API is that the content type of the transported data is RDF.

The *connection* of two services is simply a sequential execution of HTTP requests to different resources. After uploading the picture, a client can publish information about the picture upload to the user's micro blog timeline by POSTing the information to his timeline. The hypermedia constraint requires that the resource representation contains links to other resources a client can interact with. This enables a client to discover the next steps in his interaction toward his goal. If in our example the picture host cooperates with the micro blog (e.g., because he is owned by the same company), he can include a link to the timeline resource in his response to the picture upload. A client can be implemented to automatically POST information to the provided link, thus realising a more flexible sequence of HTTP requests: If the picture host decides to outsource the micro blog, he can simply include a different link in his responses without breaking the interaction with existing clients. Figure 1 illustrates the client server interactions. Linked APIs automatically fulfill the hypermedia constraint, due to the inherent design of RDF. The communicated RDF data contains URIs that identify resources; the representation of these resources can be retrieved via HTTP GET. However, to enable a sophisticated interaction, links to resources should be included, which allow the application of other HTTP verbs. In a Linked API a client can not only identify links, but use the graph pattern-based descriptions to infer the effect an interaction has: the graph patterns allow for a description of what should be communicated and provide an understanding of how the interaction influences a given resource. Therefore the descriptions support a client in making automated interactions in a series of HTTP requests. The descriptions provide a clear understanding of the effect an interaction with a resource has and enable decisions on client side during runtime.

Substitution of a REST service primarily means an exchange of the resources (and therefore URIs) a client interacts with. E.g., to submit the information about the picture upload to a different micro blog timeline, the client has to send a similar HTTP POST request using another URI that identifies the new micro blog timeline. If the new resource (i.e., timeline) understands different formats than the original resource, a client in a traditional RESTful interaction would have to use another content type processor. In an Linked API a client is relieved from choosing appropriate content type processors: since the resource representations are in RDF, client and server can use schema information of the used RDF vocabularies to understand the communicated data. Furthermore, the published graph patterns allow for clients to infer the required structure of the submitted RDF data.

4 Discussion

Comparing the WS-* stack with REST architectures shows that WS-* and its semantic extensions aim to establish a service model, that covers as many aspects of services as possible (e.g., design, interaction, discovery, process description). REST only provides a lean framework based on a constrained set of uniform operations, thus focussing on the core aspects of services. The diversified service model of WS-* has lead to an intricate design and interaction process, based on a variety of standards. Standards are interchangeable due to a layered service model. Many of these standards focus on aspects that are not directly addressed by HTTP-based REST services (e.g., policy, security).

The WS-* approach builds on the availability of tools to master much of its complexity. One standard class of tools for WS-* are those that automatically create WSDL descriptions from source code and vice versa. Such tools make it very easy to provide and consume services for developers, but also lead to service interfaces that directly reflect the internal implementation, because it is very easy to expose all methods of a class. The tools can also create code that can be called like local code but encapsulate a call to the service. Changes in the implementation of the service will often also result in changes in the service interface, although the basic functionality is unchanged or only extended. Changes in the service interface will break the clients, which include compiled code that was generated based on old versions of the WSDL file.

Semantic Web services inherit the benefits and complexity of WS-* and excel in the increased degree of automation, e.g., in service discovery, mediation, and composition However, the development and maintenance of necessary complex tool suites for the SWS approaches suffers from missing industry backing. Furthermore, the manual provision of lifting and lowering can be very elaborate and hampers the flexibility of the SWS vision.

REST aims at horizontal scalability by focussing on an extensible service model that is integrated in the Web, rather than just using the Web as transportation layer for messages. In a RESTful architecture *loose coupling* is the primacy of service design to enable a dynamically interoperating ecosystem of services in the Web. Services realise loose coupling by constraining the allowed operations to a uniform set of HTTP verbs (in contrast to arbitrary operations in WS-*) to manipulate resources. Also the hypermedia constraint contributes to the loose coupling by providing high flexibility in the face of server-side changes: links to other resources, which are served by the server, control the client behaviour at runtime. The interaction is not determined by something that was decided in the past (like in a BPEL description for WS-* services).

There are no explicit mechanisms to support information integrity to prevent tight coupling between client and server. REST services are ambivalent toward dangling pointers. Only the response status codes allow a client to deal with moved or deleted resources.

Performance is a drawback of REST services, due to the stateless synchronous request-response pattern. A server is required to forget about a client after a response (sessions are not allowed), therefore every request must contain all necessary information. However, this issue can be mitigated by caching of resource representations.

Application-specific content types of various service providers require a client to use specific content type processors. Even though this design fosters loose coupling it results in heterogeneity of services, thus hampering service interoperability. Linked APIs improve on service interoperability by leveraging RDF as self-descriptive content type with shared understanding across domains.

Furthermore the use of RDF as state representation format allows to abandon the introduction of an additional semantic layer in the service model, i.e., the linking of syntactical descriptions to ontological concepts in the semantic extensions of WS-* (cf. Section 2). The hypermedia constraint allows a client to see the next possible steps in his interaction with a service. Linked APIs allow additionally to infer the consequences of an interaction by providing graph pattern-based descriptions.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
2. Blaha, M.: Data modeling is important for soa. In: Trujillo, J., Dobbie, G., Kangassalo, H., Hartmann, S., Kirchberg, M., Rossi, M., Reinhartz-Berger, I., Zimányi, E., Frasincar, F. (eds.) Advances in Conceptual Modeling – Applications and Challenges, Lecture Notes in Computer Science, vol. 6413, pp. 255–264. Springer Berlin / Heidelberg (2010), `http://dx.doi.org/10.1007/978-3-642-16385-2_32`, 10.1007/978-3-642-16385-2_32
3. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language wsml: An overview. In: ESWC. pp. 590–604 (2006)
4. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C., Norton, B.: Irs-iii: A broker for semantic web services based applications. In: In proceedings of the 5 th International Semantic Web Conference (ISWC 2006. pp. 201–214 (2006)
5. Cardoso, J., Sheth, A.: Semantic Web Services, Processes and Applications. Springer (August 2006)
6. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (November 2006)
7. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
8. Krummenacher, R., Norton, B., Marte, A.: Towards Linked Open Services. In: 3rd Future Internet Symposium (September 2010)
9. Nitzsche, J., Van Lessen, T., Karastoyanova, D., Leymann, F.: Bpel for semantic web services (bpel4sws). In: Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems - Volume Part I. pp. 179–188. OTM'07, Springer-Verlag, Berlin, Heidelberg (2007), `http://dl.acm.org/citation.cfm?id=1780909.1780953`
10. Pedrinaci, C., Domingue, J., Krummenacher, R.: Services and the Web of Data: An Unexploited Symbiosis. In: AAAI Spring Symposium (March 2010)
11. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (May 2007)
12. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1(1), 77–106 (2005)
13. Sbodio, M., Moulin, C.: SPARQL as an Expression Language for OWL-S. In: Workshop on OWL-S: Experiences and Directions at 4th European Semantic Web Conference (June 2007)
14. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: Proceedings of 8th Extended Semantic Web Conference, ESWC 2011. pp. 170–184 (2011)
15. Studer, R., Grimm, S., Abecker, A. (eds.): Semantic Web Services: Concepts, Technologies, and Applications. Springer (June 2007)
16. Verborgh, R., Steiner, T., Deursen, D.V., de Walle, R.V., Valls, J.G.: Efficient runtime service discovery and consumption with hyperlinked restdesc. In: The 7th International Conference on Next Generation Web Services Practices (NWeSP 2011 (2011)
17. Vitvar, T., Kopecky, J., Viskova, J., Fensel, D.: WSMO-Lite Annotations for Web Services. In: 5th European Semantic Web Conferences. pp. 674–689 (June 2008)