What are you talking about?

Erik Wilde
School of Information, UC Berkeley

Abstract

While services are widely regarded as an important new concept in IT architecture, so far there is no consolidated concept about the exact meaning of the term "service orientation". While there are many problems which are simply problems of certain technical decisions, other areas are more fundamental and lead to different perspectives and eventually implementations of service oriented systems. We argue that the current emphasis of service orientation as a collection of interface descriptions misses the critical point of services, which is that they revolve around resources. With a more resource-centered approach, the investment into a service oriented architecture can be made much more promising, because the resource-centered approach is better suited for the design of loosely coupled systems than the current interface-based approach.

1 Introduction

It is generally agreed that a *Service Oriented Architecture (SOA)* is an improvement over more traditional approaches for building loosely coupled heterogeneous IT systems, but there is little agreement on what that actually means. One extreme is the idea that using newer software engineering tools which publish existing APIs using the *Web Service Description Language (WSDL)* [5] already delivers a substantial benefit. The other extreme is the approach to model SOA using modeling tools which favor a top-down approach, thereby often losing the ability to accommodate services from outside of this rather closed modeling world.

In this paper, we argue that an important and often overlooked facet of SOA is the question of *resources* which are handled, and that agreement on these resources (their identification, their naming, their data model, the actions which can be performed on them) should be a central building block of a well-designed SOA. We also argue that the term "SOA" maybe a misnomer in the first place, because it assumes that this is something which can be designed starting from scratch, and with a single point of control. It is important to realize that IT systems are an existing and growing system of artifacts and tools for handling them, and that the emphasis should be on providing a cooperative environment for integrating various resources from a variety of autonomous sources, rather than on building new artifacts.

Our approach relies on the concept of object-orientation, and on the ideas of *Representational State Transfer (REST)* [10], which emphasizes the need to focus on resources as the central building block for loosely coupled systems, and on *Document Engineering* [11], which emphasized the need to model business transactions as a series of document exchanges, where the documents are representations of artifacts.

2 Web Services

Web services and SOA as they are regarded today mainly revolve around SOAP-based [14] APIs, described by WSDL [5] interface descriptions, which are then stored in a UDDI repository. There are several problems with this model:

- SOAP is a format that conveniently bridges across different programming environments, thus eliminating the need for other language-neutral interface technologies. Because of the large demand for Web services, all major software vendors offer fully automated solutions for wrapping existing APIs with SOAP marshaling and unmarshaling code. This opens APIs to a larger set of potential users using a standardized exchange format, but apart from that is no substantial change in the IT landscape.

- WSDL in fact is not a language for describing services, it is a language for describing the interfaces of services. This is an important distinction, because this means that WSDL focuses on the traditional way system descriptions inspired by software engineering are created: interfaces are described in terms of input and output parameters. The essence of what a service actually does, working with resources, and how these resources are represented, is only a by-product of the interface description.

- UDDI is an interface repository, collecting WSDL descriptions. So rather than registering services, users first store and then can search for interfaces. The weakness of WSDL's interface-oriented approach is thus mirrored in UDDI and its focus on WSDL descriptions. Additional semantics can be used to improve the descriptions of registered services, but this is not standardized and thus does not help when crossing organizational boundaries.

The emphasis on interface descriptions, which is reflected in all these technologies, has its roots in the history of SOAP and the standards later following it, and this history also is an interesting lesson in how technology is shaped by the ways the thinking of those advancing it is structured.

SOAP was designed in an effort to use XML as a wire format for calling remote procedures. So in a way SOAP was simply triggered by the observation that XML had become the universally accepted format for structured data, and that enabling RPC-like technologies based on XML would be a better solution than using less popular wire formats. SOAP's idea had nothing to do with loosely coupled systems or any other approach to build large heterogeneous systems, it was simply a widely supported wire format to be wrapped around APIs.

On the other hand, the Web, the by most successful loosely coupled IT system ever built[1], did not really contribute anything to SOAP, except for the wire format, and then later, when XML Schema [17] was created, the type system for describing the structures going over the wire. WSDL was conceived as a language to use the type system of XML Schema and describe the operations of a software component in terms of their input and output parameters.

The competing approach for how the design and implementation of Web services could be approached is the resource-oriented approach inspired by the Web's architecture, and it starts from a different perspective. Rather than being created from the perspective of tightly coupled systems traditionally built by software engineering, it has been created from the perspective of the loosely coupled view of information systems design.

3 Resources

API-oriented SOAs can be directly derived from existing software by simply generating WSDL descriptions and SOAP stubs for existing APIs. This ability to "implement SOA" with the click of an updated IDE is certainly appealing, but it is questionable (and in our view unlikely) that this kind of "wrap your APIs in XML" can deliver substantial advantages. The main problem is that the API-oriented approach always relies on the implicit assumption that there is a shared underlying data model. Without that, developers would not be able to understand an API and use it properly. This is true as long as a SOA is developed and deployed in a centrally controlled environment (and this is the reason why API-oriented SOA approaches do provide initial benefits),

Looking at the current SOA wave and the emphasis on wrapping existing (or slightly consolidated) APIs in SOAP it is clear and already obvious that this first wave of SOA-like approaches produces a certain amount of success. By now having XML-based APIs rather than language-specific APIs, software components can be connected in ways which were previously impossible. However, the design-based tight coupling of the API-oriented software engineering approach is not really affected by this change, so after the first wave of success, the fundamental design problems (hidden data models, opaque services) will surface again.

However, the API-oriented SOA approach does not adapt well to an environment of multiple autonomous peers, where each peer may have his own data model, and there is no implicitly shared data model to start with. In such a scenario, the problems of the API-oriented approach become apparent, because in the absence of an implicitly shared model, the API alone does not provide sufficient information to use a service. In fact, the most important question about the service, which is the question about the resources affected through or managed by that service, is not answered in a useful way.

Resource-oriented SOAs, on the other hand, always require additional efforts, because they need planning and implementation efforts for naming and handling resources. While the effort to plan and implement this often is acceptable, it almost always is more effort than the "generate your WSDL" approach of API-oriented SOAs.

4 Resource-Orientation

Successful modeling and implementation depends on identifying the involved resources, and the actions which can be performed involving these resources. Object-oriented analysis [7] (and the earlier structured programming developments in that direction) moved the focus of modeling and subsequent implementation from APIs to objects, introducing the concept of *encapsulation* as one of the core concepts of object-orientation [1]. Encapsulation demands that implementation details should be hidden through an interface which only exposes methods.

In a loosely coupled world, for example one with multiple autonomous peers in a service scenario, encapsula-

[1] The interesting fact is that the Web was never really "built", it emerged because of the REST design and the resulting loose coupling, and the systems implementing it evolved in ways which never would have been predictable in the first years of the Web. The Web is successful because it did never have APIs.

tion cannot be enforced in the same way as it can be in a homogeneous software environment. The reason for that is that while the homogeneous environment assumes there is a data model in the implementation and hides it behind methods to achieve a higher level of abstraction, there is no such implicit shared model in loosely coupled scenarios. Instead, when interacting, peers must deal with the incompatibility of their models and maybe even their formats, and must make conscious efforts to agree on a shared model and a shared format. Often, this agreement will only need to cover a subset of the complete data models of both peers, which is the subset that is relevant for that particular service interaction.

For example, when two companies want to exchange personnel records, they need to agree on the model of these records, and they have to make an effort to create a shared model of that subset of the records which is relevant. This is something which cannot be automated easily, and which more importantly cannot be done by simply matching interfaces. Only if the peers use the same model, no additional agreement is necessary. The *Universal Business Language (UBL)* [4] is an attempt to define a number of useful models which help to avoid the need for explicit agreement, but this approach of a library of popular models naturally is limited to a rather small set of popular resource types for a specific application domain.

For a more robust approach which also works for resources which do not use a priori matching models, the resource description must be made available as the first class citizen of such a service description. The currently favored approach to Web services (as described in the following section) does not provide reasonable access to the descriptions of the resources being affected by a service, and thus is not well-suited to deal with scenarios without a priori matching models. Instead, we favor an approach described in more detail in Section 7, which is to make XML Schema better accessible to programmers [13], and to make this accessible schema available in an easily usable way in a popular XML technology [18].

5 SOA and SOAP

The biggest problem with the SOAP-based approach to SOA is that it hides the data model behind an interface which is not designed to make available any information about the underlying data model of the service. What WSDL does is *not* describing the service (even though its name implies this), it simply describes the interface. The interface might expose some fragments of the resources handled by the service, but these appear only as second-level constructs as part of the SOAP operations. The required *sharing of a common model* between service providers and consumers is thus obscured by the mere *sharing of an interface* through which data is being exchanged that rarely is described in a semantically meaningful way.

WSDL descriptions revolve around the abstract operations of a service, which are then bound to a concrete service. WSDL also includes a part describing types, which are used to describe the message structures (which are used in the operations). However, this type part of WSDL is treated as something auxiliary to the interface description. Second and more importantly, as pointed out by Prescod [16], the design of WSDL makes it almost impossible to use URIs as resource identifications (which is the core idea of the Web's architecture [12]). Instead, in the SOAP/WSDL model, URIs become service endpoints, where one URI is the access point through which all interactions with the service are managed.

Neither SOAP nor any of the auxiliary technologies define a standard for addressing resources at all. This means that every service oriented scenario has to come up with its own scheme of how resources are addressed. This turns the fundamental principle of the Web, the fact that everything revolves around resources, into an issue which is not even addressed by the SOAP/WSDL approach.

The SOAP/WSDL approach therefore reduces the Web to a transport infrastructure, basically ignoring the fact that the Web is not just that, a transport infrastructure, but an application-level architecture for an information system. Because the SOAP/WSDL approach does treat the Web as a transport infrastructure, it needs to recreate a lot of functionality which typically sits on top of a pure transport service. The rapidly growing SOAP protocol stack is an indication of the fact that instead of being "Web Services" which are a part of the Web, they are "Web Services" sitting on top of the Web.

Historically, the somewhat curious "Web Service" name used for a technology which ignores a lot of the Web's architecture has developed out of the observation of application programmers (which did not have anything to do with the Web as an information system or Web contents at all) that (1) XML was becoming a widely accepted format for structured data with a lot of interesting and powerful technologies and tools around it, and (2) the easiest way to get through firewalls when using Internet connections was to use port 80, the standard port for Web servers, which in many cases was the only open port in corporate firewalls.

Starting with that observation (HTTP is useful as a transport service, XML is useful as a wire format), "Web Services" were named like this because they used Web technologies. It was not so much a question of whether they used them in the way they were supposed to be used, and in fact, for many newcomers to the "Web Services" field, it is hard to grasp at first that even though they are called "Web Services", they have surprisingly little in common with the Web.

SOAP quickly gained popularity, because turning any API into a SOAP-based Web service can be fully automated. Vendors of software development environments quickly provided support for that kind of code generation, and every major programming environment today provides support for taking interface definitions in some programming language, and generating WSDL descriptions and SOAP stubs out of these. So from a developer point of view, SOAP/WSDL-based Web services can be provided without any additional implementation effort, which is an attractive proposition.

In contrast, a resource-centered approach always requires planning and additional implementation effort, because REST demands that resources are identified by URIs and designing a URI-based namespace can be a non-trivial task. This means that resource-centered services require much more effort when being designed and implemented, in particular when compared to the "single-click" approach of just publishing APIs.

The question then is why should users choose resource-orientation for SOAs, when SOAP is so much easier to build? The problem here is that most SOA scenarios in fact are closed world scenarios (i.e., a shared model is taken for granted), in which the real problems of the API-centered approach simply do not appear. And as long as a service is only intended to be used in-house and the implicit shared data model of the API-centered approach can be safely assumed, there will be no problems building and using API-centered SOAs. However, the increasingly fragmented and distributed nature of business processes will more and more lead to situations where the closed world assumption of the API-centered approach does not hold anymore, and then the well-known problems of interface incompatibilities (what is the data model which users need to know to use a particular service?) will resurface.

6 Top-Down vs. Bottom-Up

Service orientation is situated in between two different areas of applied computer science, one area being the business modeling which traditionally focuses on high-level issues and employs top-down approaches such as the *Model-Driven Architecture (MDA)*. On the other hand, the promise of loose coupling often is implemented on the fairly technical level of building mashups or integrating services available from various sources; this area often operates very much bottom-up. There currently is a gap between these areas (discussed in more detail in Section 7) which not only leads to possible long-term problems, but also makes it hard to realize the full potential of service orientation.

6.1 Top-Down

Traditionally, large-scale IT systems are planned and built using rigorous top-down approaches, in most cases building on modeling methodologies such as *Model Driven Engineering (MDE)*.

The term "Service Oriented Architecture" already carries rather strong undertones of the idea of top-down modeling from the scratch. Architecture usually is a discipline which starts with a clean slate, takes a set of constraints and requirements, and applies a set of rules and methods to create a result which complies with the initial specifications. However, we think that instead of evoking the idea of a clean slate, the real challenge in service orientation is to build something that integrates a large set of existing services which the designer has no control over. This means the real challenge and goal of service orientation is to gain the ability to accept and integrate what is already there, and build something which fits into and interacts with other players in that landscape.

Starting from the idea of an "architecture", it is not a surprise that the *Model-Driven Architecture (MDA)* approach often is perceived as a good complement to SOA: While SOA is regarded as the goal, MDA is regarded as the set of methods to achieve that goal. Given the top-down orientation of both MDA and an API-oriented SOA, it is true that MDA can be regarded as a way to model and implement a SOA system.

6.2 Bottom-Up

While the area of software engineering developed increasingly top-heavy approaches, the area of XML technologies has grown bottom-up. First perceived as a convenient wire format, XML has grown as a response to developer demand. XML Schema [17] introduced the datatype library that application programmers wanted, and the recently finished *XML Query (XQuery)* [3] family of specifications establishes XML as a full-scale database model, which may even become a serious threat to the less expressive relational model.

However, even though XML has now progressed to the point that it not only has a well-defined physical model (XML documents), but also a well-defined and type-oriented logical model, the *XQuery 1.0 and XPath 2.0 Data Model (XDM)* [9], there still is no conceptual model for XML [15]. There is some ongoing research, but surprisingly little, and there is neither an agreed upon de-facto standard, nor is there active work in the relevant standardization groups. The reason why there is no conceptual model for XML and only little initiative to change that is the gap between the bottom-up development of XML (which has now reached a point where, from the programming

point of view, the necessary technologies are available), and the top-down modeling and design of IT systems, which still largely ignores XML *as a conceptual data model* and sees it as a pure encoding of structured data.

7 Bridging the Gap

As stated in the previous section, there currently is a gap between the top-down approach of the usual SOA approaches, and the bottom-up evolution of the XML technology landscape. The main reason for that gap is that the top-down approach mainly has been pushed by a community with a strong background in software engineering, while the bottom-up development has grown out of the Web technology area. The main problem is that because of this gap, there is no easy way to link data model concepts from the modeling world with interface descriptions for Web services.

A rather heavy-weight approach to solve that problem is championed by the *Semantic Web* community, which is attempting to augment WSDL with more model-oriented descriptions [8] (in this case these are of course RDF descriptions), so that services can be described in a more semantic way than by just describing the interface. The general approach of trying to enrich WSDL so that it becomes more expressive is going into the right direction, but we see two major problems with this approach:

- WSDL still relies on the API-centered approach, so while the semantic annotations make it possible to establish connections between the interface description and a model description, the overall approach still is not resource-oriented.

- Semantic Web technologies have the tendency to be built with a more top-down mind set, where the resulting architectures work well in a setting where all participants are supporting the same set of technologies, whereas it is hard to make any use of the information without supporting the full tool set.

So instead of relying on the Web service users to start using Semantic Web technologies, we envision a more bottom-up approach, which should finally bring the XML and the modeling world together. Figure 1 shows the current state of the art in handling XML, and the development that we see for the future.

XPath was created in the early days of XML and provided a convenient way of selecting parts of an XML document. However, with the introduction of the concept of XML types with XML Schema, XML 1.0 proved to be insufficient, because it cannot access the type information associated with an XML document. XPath 2.0 [2] addressed

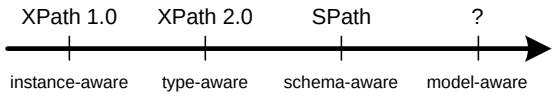

Figure 1. Scope of XML Technologies

that issue by adding type awareness, which means that information can now be selected based on type information as well. However, the functionality of XPath 2.0 is limited to XML documents only, which means that schemas cannot be accessed.

The *XML Schema Path Language (SPath)* [18] is an extension of XPath 2.0 which provides schema access as well. The idea of SPath is to provide the recipients of XML documents with a better way to inspect the associated schema, thereby catering for use cases such as versioning of XML (which is important for an independently evolving landscape of Web services), or schema-oriented applications, which for example may generate interfaces for schemas describing the documents of remote Web services.

As a final step on that scale, it is possible to envision a technology which goes even beyond the schema and enables users to find a connection to the underlying model. Again, semantic Web technologies provide a possible solution, but they are heavy-weight, and they closely couple the semantic annotation with the language to describe the model. We believe that a more light-weight approach would have better chances to cater for the variety of modeling approaches which users might want to employ. This of course carries the risk that a recipient of an XML documents find a model annotation but cannot understand the model, but this is one of the strengths of the Web, to allow diversity and provide an environment where this diversity can develop.[2] Based on model annotation, the idea of model mapping [19] (in case of model compatibility, of course) could be the final step towards a Web which closes the gap between the bottom-up XML technologies and the top-down modeling approaches.

Finally, one problem with the current state of the art surrounding the REST approach is that there never evolved something like a "WSDL for REST". For the "Web REST service", no interface description was necessary, because that is what regular Web servers do. Most other REST services which are used tend to define a small set of resource URIs, and often do this textually. For small REST services this may be acceptable, but it maybe harder to read and understand than a well-defined formal description, and more importantly, only formal descriptions allow to write tools acting on these descriptions.

The new version of WSDL [6] also provides support for

[2] Using HTTP's *content negotiation*, peers could even try to negotiate a common model format, using available Web features to improve compatibility in a heterogeneous environment.

describing REST services, and it remains to be seen how well that new part of the language will be received and how much it will be used. Because of its heritage in the API-centered world of Web services, the WSDL family of specifications now supports both worlds and as a result has grown to a rather complex set of recommendations. How well this will work in an open setting where the choice of technology cannot be mandated and people generally have a tendency to choose the simplest tools which do the job remains to be seen.

8 Conclusions

The short-term benefits of opening up an API to a wider set of application by turning it from a language-specific API to a WSDL-described SOAP API overshadow the fundamental problems of API-centered approaches. While the current benefits of API-centered approaches are real, the lack of a more resource-centered approach will probably lead to a backlash, once the first SOA systems start to experience the same API-caused problems which they were facing before, only this time across programming language barriers.

From the perspective of vendors for IT infrastructure, there is little incentive to make the switch from API-orientation to resource-orientation. In fact, it may be regarded as a risk from the vendor side if the data model of an IT system has to be made as open as it has to be for a truly resource-centered approach. Instead, keeping the data model hidden behind the APIs and making these available through a different interface technology, makes more business sense for vendors.

The current gap between the XML landscape, which evolved from the bottom-up, and the SOA idea, which grew out of the world of traditional software engineering, makes it currently hard to build resource-centered SOA systems. However, we believe that the increasing fragmentation of IT landscapes makes it inevitable that IT systems are built more around the resources they are handling, and less around the methods they provide to handle hidden data instances.

References

[1] D. J. Armstrong. The Quarks of Object-Oriented Development. *Communications of the ACM*, 49(2):123–128, February 2006.

[2] A. Berglund, S. Boag, D. D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. World Wide Web Consortium, Recommendation REC-xpath20-20070123, January 2007.

[3] S. Boag, D. D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Recommendation REC-xquery-20070123, January 2007.

[4] J. Bosak, T. McGrath, and G. K. Holman. Universal Business Language v2.0. Organization for the Advancement of Structured Information Standards (OASIS), Standard, December 2006.

[5] R. Chinnici, M. Gudgin, J.-J. Moreau, and S. Weerawarana. Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl12-20030611, June 2003.

[6] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl20-20070326, March 2007.

[7] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Prentice-Hall, Upper Saddle River, New Jersey, 2nd edition, October 1991.

[8] J. Farrell and H. Lausen. Semantic Annotations for WSDL. World Wide Web Consortium, Working Draft WD-sawsdl-20070410, April 2007.

[9] M. F. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model (XDM). World Wide Web Consortium, Recommendation REC-xpath-datamodel-20070123, January 2007.

[10] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[11] R. J. Glushko and T. McGrath. *Document Engineering*. The MIT Press, Cambridge, Massachusetts, August 2005.

[12] I. Jacobs and N. Walsh. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.

[13] F. Michel. Opening XML Schema's Data Model to XPath 2.0. Technical Report TIK Report No. 264, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, November 2006.

[14] N. Mitra. SOAP Version 1.2 Part 0: Primer. World Wide Web Consortium, Recommendation REC-soap12-part0-20030624, June 2003.

[15] S. Mohan and A. Sengupta. Conceptual Modeling for XML — A Myth or a Reality? In Z. Ma, editor, *Database Modeling for Industrial Data Management: Emerging Technologies and Applications*, chapter X, pages 293–322. Idea Group Inc., Hershey, Pennsylvania, December 2005.

[16] P. Prescod. Roots of the REST/SOAP Debate. In *Proceedings of 2002 Extreme Markup Languages Conference*, Montréal, Canada, August 2002.

[17] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[18] E. Wilde and F. Michel. SPath: A Path Language for XML Schema. Technical Report UCB iSchool Report 2007-001, School of Information, UC Berkeley, Berkeley, California, February 2007.

[19] E. Wilde. Model Mapping in XML-Oriented Environments. Technical Report TIK Report No. 257, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, July 2006.