

Web Engineering Revisited

Erik Wilde
School of Information
UC Berkeley
dret@berkeley.edu

Martin Gaedke
Faculty of Computer Science
Chemnitz University of Technology
martin.gaedke@informatik.tu-chemnitz.de

Abstract

Web Engineering has become one of the core disciplines for building Web-oriented applications. This paper proposes to reposition Web engineering to be more specific to what the Web is, by which we mean not only an interface technology, but an information system, into which Web-oriented applications have to be embedded. More traditional Web applications often are just user interfaces to data silos, whereas the last years have shown that well-designed Web-oriented applications can essentially start with no data, and derive all their value from being open and attracting users on a large scale. We propose “Web Engineering 2.0” to not focus anymore on *how to engineer for the Web*, but *how to engineer the Web*. Such an approach to Web engineering not only leads to a more disciplined way of engineering the Web, it also allows computer science to better integrate the special properties of the Web, most importantly the loosely coupled nature of the Web, and the importance of the social systems driving the Web.

Keywords: Web engineering, REST, SOA, Loose coupling, Web architecture

1. INTRODUCTION

The Web has become one of the most transforming computer-based technologies ever invented, and has driven a number of important research and engineering directions in the almost 20 years since its invention in 1989. As the most prestigious conference in the Web community, the WWW conferences is a good indicator of popular Web themes, with a steady flow of papers from searching and ranking and more general information retrieval techniques. Other themes are much more influenced by trends and have been or maybe will be rather short-lived. There are surprisingly little consistent themes in the conference topics, one of course being everything search-related, and the other being *Web Engineering*, mostly interpreted as looking at methods and tools for building Web-oriented applications.

The last couple of years have seen many developments in computer science and IT technologies gravitating towards the Web. The Web is increasingly perceived as “the platform” for future developments in many areas, but the way in which this phrase is interpreted is very different. There are probably three major viewpoints of this “platform concept”, the first is to see Web-based user interfaces as the platform, leading to the rather simple idea of providing users with Web-based interfaces to applications (instead of building standalone or platform-specific client/server-applications). The second viewpoint is seeing the browser itself as a platform for development, where the increasing richness of this client-side platform allows new classes of applications, sometimes shifting almost all application logic to the browser. The third viewpoint is to see Web-oriented client-side environments as the development platform; this approach often is referred to as a *Rich Internet Application (RIA)*.

All three approaches attract a lot of interest and are based on different assumptions and create different constraints in many areas, such as security, performance, reusability, robustness,

portability, usability, and accessibility. But all of these questions are mainly questions about *how to engineer Web-oriented applications*. What this paper suggests is to pay more attention to an area which has received much less interest in research, namely *how to engineer the Web*. Many of the developments that shaped the triadic world view of the Web described above were not so much planned and engineered, but they were somehow introduced and then developed evolutionary. This evolutionary process is a good thing and should not be changed radically, but this paper argues that it would be possible to better understand and control that process, and thereby to better plan and develop for the Web's future.

Today, *Web Engineering* is largely understood as *how to engineer for the Web* instead of *how to engineer the Web*, and this looks at Web engineering as a specialized variant of software engineering, taking into account the special factors of the Web as an application development environment. A much wider view has been introduced with the concept of *Web Science* [1], which focuses on an interdisciplinary approach to better understand all kinds of phenomena somehow associated with the Web.

Our proposal is to extend the reach of the current Web engineering to a perspective where Web-oriented applications are no longer perceived as using the Web as a platform, but where the ultimate goal is to make them part of the Web, following its architectural principles, and benefiting from its adaptivity, ubiquity, accessibility, and flexibility. Section 5 describes this extended idea of Web engineering in greater detail, referring to it as *Web Engineering 2.0*. We propose to rethink Web engineering as a discipline which strays further from traditional top-down ideas of software engineering, and focuses more on the complete Web as the system into which an application has to be embedded.

For example, it is surprising to see that not a single paper in the recent Web engineering conferences (ICWE) was focusing on syndication and how to further develop the landscape of systematically repurposing content. Following a more systemic approach, looking at syndication, its progress over the last years, and the open problems, it would follow that Web engineering should look at improving syndication for turning the Web into a better system for content publishing, aggregation, filtering, and repurposing. However, the current application-specific focus of Web engineering excludes this kind of viewpoint. The separation of the Web as a system and Web applications and individual entities developed in isolation prevents Web engineering as a discipline to reach its full potential as the premier driver of technical innovation of the Web.

2. WEB ARCHITECTURE

One of the reasons why the Web has been so much more successful than all previous approaches at building large-scale information systems was (and still is) its simplicity and evolutionary development. While there is an *Architecture of the World Wide Web* [7], this is mostly a post-hoc rationalization of some of the developments and technologies, and only partly drives today's development of the Web.

Historically, a surprising number of key developments on the Web happened (and in some disciplines still happen) with surprisingly little influence from mainstream computer science research. In the beginning, the Web was dismissed as being much less sophisticated than the state-of-the-art in academic information systems, and there still is a certain tension between the way academia develops, incentivizes and assesses research, and the way how the Web is developing. The following list is not complete, but is intended as a brief overview of the differences between typical academia approaches, and how the Web works:

- *Revolutionary vs. Evolutionary*: Because of its size and pervasiveness, the Web must develop evolutionary. In contrast, academia typically tackles idealized problems, where it is easier to solve the core parts of a problem, without spending too much effort on dealing with compatibility and versioning issues. On the Web, these problems often are the core problems, and they are particularly hard because of the lack of any centralized control.
- *Completeness vs. Simplicity*: Typical computer science approaches try to solve an idealized problem completely. A typical Web approach solves a real-world problem (with all constraints

caused by evolutionary development) in a way which is good enough for most scenarios. This model of simpler solutions not only allows the technologies to be simpler, it also allows them to be more extensible so that unforeseen uses are possible. Given sufficient success, a technology may then evolve to a more mature version, mostly taking into account the feedback from how the previous version was used.¹

- *Isolation vs. Systemic*: Computer science typically uses the standard scientific approach of isolating a problem, and then studying it in its idealized form. The Web, on the other hand, is a global system with many unknown interdependencies between technical and non-technical components of that system. Looking at Web-oriented applications in isolation therefore ignores essential features of the Web environment. Web Science and *Service Science* [16] address this fact and conclude that a lot of the value generated through the Web is co-creation, where the whole Web as a system and as an environment has to be considered to understand how to design and build services.

While Web architecture oftentimes strives for simplicity, this does not mean that it is simple to develop new ideas and technologies. On the contrary, the added constraint of striving for a simple solution can make a problem harder to solve than if the proposed solution has no limitations on complexity. One of the main challenges of engineering the Web lies in identifying the most promising ideas for improvements, and building a simple solution for them. For example, technically speaking, a large part of the Web 2.0 wave can be attributed to a simple standardized function in JavaScript, which allows scripts to communicate with the server. This simple solution has many limitations and side-effects, but it has served as one important building block for an astonishing number of new developments.

One distinguishing feature of the Web is that its design allows errors to happen and provides an environment in which the main goal is not to prevent errors, but to design the system to be able to cope with them. Another important feature is loose coupling, which is a frequently used term with little agreement on what it actually means. Section 4 examines the concept in greater detail, whereas Section 3 first describes how Web Engineering had and partly still has to evolve from the pretty standard top-down approach of software engineering, to a discipline with a wider scope, so that it can better support the design and implementation of successful Web applications.

3. SERVICES ORIENTED ARCHITECTURE

The last years have seen an increase in complexity in the typical enterprise IT landscape [15], a growing need to open up enterprise services to better collaborate with business partners, and a faster pace at which this has to be accomplished. Traditional IT strategies and architectures and their approach of top-down design and implementation have problems to adapt to the increasingly fragmented and fast-changing landscape of large organizations. Various proposals for middleware architectures have tried to make integration easier and more agile, and one of these approaches is the *Service Oriented Architecture (SOA)*. The goal of SOA is to identify and isolate services which can be provided as reusable components to other services. SOA's approach is to reflect organizational structures, where services in organizations are provided because they are needed to perform the business functions of the organization.

However, SOA by its very nature still is an integration approach, trying to solve the integration problem better than previous approaches. Integration can be referred to as an "architectural style", characterized by the attempt to build abstractions that make a distributed and heterogeneous system look like a centralized and homogenous system, at least in some regards. This integration approach still is the approach favored for many organizational IT landscapes, mostly those in enterprise or organizational settings. The Web, however, follows a different architectural style, which is called *Representational State Transfer (REST)* [3] and focuses on loose coupling (discussed in more detail in Section 4) instead of integration.

¹HTML and CSS are excellent examples of such a development, with very simple first versions. Both technologies then matured according to real-world observations of how they are used. Interestingly, popular JavaScript libraries can serve as a good indicator what kind of functionality many people perceive to be missing in the declarative technologies, and thus extend them with procedural code. These features can then be included in future versions of the declarative languages.

There still is a considerable debate around *Web Services* and the general question of how IT systems should be architected and implemented. Oftentimes, the discussion mixes architectural and implementation issues, and fails to lead to a comparison of approaches on both the architectural and the implementation level [14]. The really challenging question in this field is whether the integration-oriented approach of SOA and other middleware-inspired approaches to massive-scale information systems is the right approach, even for setting smaller than the entire Web. It is a pretty well-accepted fact that the Web would not have been possible with a top-down integration style approach; the most challenging question facing many CIOs today is whether this lesson in massive-scale information system design does or does not apply to organizational IT systems.

The big difference lies in the question of the core tasks of IT systems. The notion of *Wikinomics* [17] (even though this term often is used in an overly enthusiastic way of painting the future) is increasingly getting traction. Regardless of how it is called, organizations today often are approaching the properties of the Web on various dimensions, such as the rate of change, the number of different system throughout the organization, the lack of complete top-down control, the necessity to interoperate with systems outside of the organizational domain, and the requirement to re-design and re-deploy services as quickly as possible.² A repositioned Web Engineering, focusing more on this essential issue of how content and services should be architected and implemented, could help to better understand the challenges and opportunities surrounding the SOA and Web services debates.

4. LOOSE COUPLING

Even though the term “loose coupling” now most frequently is associated with IT architectures, interestingly enough it originated in research of organizational structures [12] as early as 1967. Structurally, the problems which may cause loose coupling or which could be avoided or mitigated with implementing loose coupling are the same in organizations and IT systems: the tension between the efficiency and safety of an internal determinate and completely rational structure, and the necessity to integrate into an open world of uncertainty and conflicting concepts.

Loose coupling is often quoted as a property of some solution or product (and is almost always used as describing a good and desirable property), but so far there is little agreement of what exactly the term refers to. Loose coupling has a number of facets, and we will not attempt to give an exhaustive definition of the term here. But the interesting observation is that it does have implications beyond the SOA approach presented in Section 3, which — being based on an integration approach — still oftentimes assumes an explicit or implicit overall design. Loose coupling assumes that there is no such overarching design, and that peers interested in interactions have to overcome certain obstacles for successful interaction, and these obstacles are mainly caused by the absence of an overall design. The simpler it is to use services, the easier it will be for loosely coupled peers to successfully interact.

More enterprise-oriented approaches often mention contractual bindings and obligations as the one big thing that distinguishes a typical enterprise IT architecture from a Web-style architecture. This argument misses the point that nothing in the Web’s loose coupling approach prevents participants from entering contracts and then being bound by service level agreements. We believe that the current challenge in the area of Web services and loose coupling lies in finding a good way how architectures and applications can be build which can work in both ways, so that one scenario does not implicitly exclude the other scenario.

One of the big challenges of Web engineering is to move forward from the picture of the traditional software lifecycle, and recent trends in programming languages and software development demonstrate this need for more agile and more adaptable applications. The ideal Web application of the future starts as an initial version, and then evolves by being shaped by its users and the

²The last capability is increasingly noticed by *Service Science*, which among other things claims that the ability to adapt quickly to changing environments can be essential for providing successful services. As one example, the ability to make the service value chain more transparent allows service designers to better optimize the “service experience” [5].

Web environment, very likely in ways which have never been expected.³ The Web itself, on the other hand, will increasingly be shaped by this new class of adaptive applications, the recent wave of *Web 2.0* applications and the way how applications and data are increasingly mixed and remixed and reused are a small sign of future developments of the Web landscape. *End-User Development (EUD)* [4] is one of the major trends in Web 2.0 applications, and the better designed a Web application is as part of the Web, the more amenable it will be for being reused and repurposed in EUD scenarios.

5. WEB ENGINEERING 2.0

In many ways, today's intranets very often start to look like the Web, exposing the fractal nature of many evolving systems. They may have different sizes and different policies controlling them, but essentially, they are increasingly becoming little Webs. On the other hand, Web engineering so far has largely looked at how to build applications and services for the Web, with only little focus on the fact that increasingly, the distinction between *internal* and *external* should not be decided when designing and building an application, but by decisions based on the environment and requirements that are not static, but an evolving set of properties shaped by the application itself, its users, and the larger context of the Web itself.

Our vision of a repositioned Web engineering discipline of course has various associations with other visions of new areas of computer science or web-related approaches. *Web Science* [1] has a much wider perspective, and definitely would be important as an input for better understanding the evolving landscape of the Web, its users, and way how it is used. *Service Science* [16] is focused on identifying how services should be designed and implemented more systematically. Both of these disciplines have a rather large overlap, but so far no attempt has been made to somehow unify them. *Web Engineering 2.0* is still very much focused on technical questions of how to make content and services available on the Web, but more more concerned with the "Web" part of the name than its predecessor.

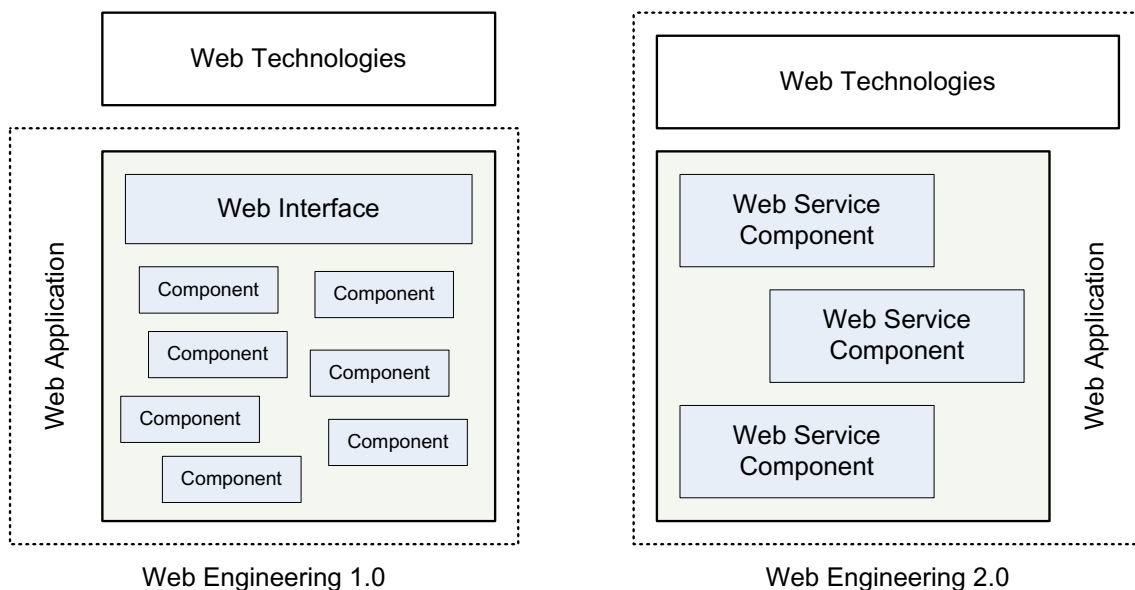


FIGURE 1: Web Engineering 1.0 and 2.0

Figure 1 shows a comparison of these two styles of Web engineering. In the left-hand side, it shows Web engineering and looking primarily at building Web applications in a disciplined way. This involves many of the well-known methods from software engineering which try to improve the process by which software is being built. Web technologies in this picture do play a role as

³One manifestation of that is the "perpetual beta" syndrome of applications on the Web, where applications never get out of beta status, assuming that at any time features might be added or removed.

constraints, but do not play a significant role as something that should also be engineered as part of the evolving environment. The right-hand side shows a different picture, where Web technologies play an integral role in Web engineering, and advancing the Web itself becomes one of the core tasks of Web engineering. Also, a Web application is no longer perceived to be one application that then will get an interface to the Web, instead the application itself is decomposed into smaller components which themselves are considered Web applications. Some of them may be designed and implemented as part of a Web engineering projects, others may be accessed that already exist on the Web, and the Web application itself opens up to be accessed as a Web service. The design of the individual components then can done using software engineering methods, but the ecosystem of cooperating components is the result of this new Web engineering.

This new Web engineering consequently has a lot to do with *Web Services*, but not in the sense of the word that has taken over this term for the past years as a field being largely concerned with questions of middleware and systems integration. Instead, Web services in this case refer to Web services based on the design principles of the Web, which means REST.

Semantic Web, largely concerned with questions of knowledge representation and reasoning, can also play a role in that scenario in that they allow Web application components to publish semantically richer information, if they wish to do so.

One of the main advantages of this new Web engineering would be to better manage the duplication of functionality that could be observed in the Web services area for the past years [8]. Instead of building a separate WS-* protocol stack layered on top of the Web as a transport system, this new perspective could re-focus Web engineering to work on improving the Web. We look at the “middleware over HTTP as transport” approach as the same transitional phenomenon that could be observed in the early days of the Internet, when people were running “SNA over IP” or “DECnet over IP”. Some legacy systems still use these kinds of architectures, but not many systems will be designed and built using them.

One of the two biggest advantages of a Web engineering discipline focusing more on the “Web” part of its name would be the fact that by engineering in a more Web-like way, engineering for evolution becomes almost an implicit goal, both in terms of engineering an application itself, and also engineering the application as a component of a bigger system. This is discussed in more detail in Section 5.1. The second advantage would be a more integral way of looking at Web applications and Web technology, and would be the best way how to work towards evolving the Web.

5.1. Engineering for Evolution

One of the main advantages of a more REST-oriented approach is the capability of services to allow *Serendipitous Reuse* [18]. This goes back to the integration vs. loose coupling approaches presented in Sections 3 and 4. The Web thrives on serendipitous reuse, and regardless of the name chosen, this simply means a lower barrier to entry.

The current trend towards *Enterprise 2.0* [9] makes the need for well-designed services even more obvious. More traditional back-end oriented architectures, such as the idea of “portals”, was based on the assumption that the goal is to build a big integration application, which integrates all required applications in the back-end, and then makes them available through a single configurable Web interface. Enterprise 2.0, on the other hand, looks at content and services being made available through Web services, so that that can be re-used, re-combined, and repurposed as required, without the need for a heavyweight integration hub. This not only allows quicker development and deployment of composite applications, it also allows the enterprise to be more agile in exploring various ways of implementing services.

5.2. Web Evolution

Engineering applications for evolution (as described in Section 5.1) helps to build applications that are better components of the Web. However, as one lesson learned from Web engineering,

applications often want to do more than is possible with the available Web technologies. There seem to be two main approaches to that problem; the first is to layer a complete set of new technologies on top of the Web, this is what happened in the Semantic Web and WS-* Web services areas. The second is something we refer to as the *Plain Web* [19] and focuses on evolutionary development of the Web, introducing as little new technologies as possible, and trying to keep those as simple as possible.

Interesting examples in that area is the field of content syndication. Content syndication is not a highly complex field, but still requires well thought through concepts to be able to use Web-scale syndication easily and effectively. The first version of a syndication format was *RSS* and after a short period of time was riddled with numerous incompatible version of the technology because of competing groups and a lack of standardization or research interest. Finally, the *Atom* [10] format and more recently the *Atom Publishing Protocol (AtomPub)* [6] emerged as better designed alternatives. Syndication plays an increasingly important role on the Web and may very well become one of the major ways of how information is being managed on the Web. AtomPub is the first large-scale REST protocol that could be used in many scenarios, the current syndication scenarios of news distribution and blogging are by no means the only scenario that could benefit from syndication.

The syndication model still is lacking a number of features. Some of these are already solved by additional specifications, but some essential features are still untackled. A first approach is available with the *Feed Item Query Language (FIQL)* [11], but most applications⁴ are building proprietary solutions. This kind of work could be an excellent way of how Web engineering could contribute to improving the Web, in this case by tackling the general question of how to build RESTful interactions around the general idea of a collection of data items.

Currently, surprisingly few Web technologies get a lot of input from computer science. Some of the more advanced areas with an established history in earlier computer science fields⁵ did receive more attention from computer science, but these fields are more the exception than the norm. We believe that computer science could and should play a bigger role in advancing the Web, and a new Web engineering could play a lead role in this.

6. CONCLUSIONS

Web Engineering 1.0 considered the evolution of applications in the following sense: taking care about the developed artifacts, i.e. designing and developing Web application artifacts that are reusable for the development of further Web applications. In *Web Engineering 2.0*, the development must take the environment and its users as an engineering principle into account, i.e. the user is not only consumer of the content provided, but also acts as a producer actively or implicitly by contributing data that enhances the Web application. In other words, Web Engineering 2.0 deals with a new kind of evolution, which is driven by the usage of the engineered and finalized Web application — enabling the user to shift from being a consumer to being a consumer and producer.

Web technologies so far has received very selective attention from computer science research; some of the advanced problems such as search and ranking, information retrieval, XML databases, the Semantic Web, and tightly coupled Web services have been covered, in many cases because there was previous work to build on, previous insights to bring to the Web, and a clear way of how established computer science could be transferred to the Web. Other areas of Web technologies which may be equally important for the Web's success have received much less attention from computer science, examples that come to mind are document representation and presentation, loosely coupled Web services, and multimedia on the Web. Web Engineering 2.0 could make the step to include some of the increasingly important aspects of Web application

⁴A popular example for this is Google's GData, which is a Google-specific extension of AtomPub that has some rudimentary query features.

⁵The most popular examples are XQuery [2] which could reuse a lot of knowledge from the database research community, and the Semantic Web and mostly OWL [13], which could reuse a lot of knowledge from the knowledge representation and reasoning research community.

development to the attention of the computer science community, and to build a closer relationship between building applications for the Web, and advancing the Web itself.

We believe that the Web can provide a number of very interesting research challenges for a more Web-oriented approach. However, the traditional incentives and expectations around computer science research often favor themes with less compatibility issues and a less heavy emphasis on simplicity of the final outcome. How to create new incentives around Web-oriented research so that it does become attractive for computer science researchers to tackle some of the currently under-researched field is probably the most challenging task in this vision of taking Web engineering one step closer to engineering the Web.

REFERENCES

- [1] TIM BERNERS-LEE, WENDY HALL, JAMES HENDLER, NIGEL SHADBOLT, and DANIEL J. WEITZNER. Creating a Science of the Web. *Science*, 313(5788):769–771, August 2006.
- [2] SCOTT BOAG, DONALD D. CHAMBERLIN, MARY F. FERNÁNDEZ, DANIELA FLORESCU, JONATHAN ROBIE, and JÉRÔME SIMÉON. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Recommendation REC-xquery-20070123, January 2007.
- [3] ROY T. FIELDING and RICHARD N. TAYLOR. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.
- [4] G. FISCHER, E. GIACCARDI, Y. YE, A. G. SUTCLIFFE, and N. MEHANDJIEV. Meta-Design: A Manifesto for End-User Development. *Communications of the ACM*, 47(9):33–37, September 2004.
- [5] ROBERT J. GLUSHKO and LINDSAY TABAS. Bridging the "Front Stage" and "Back Stage" in Service System Design. In *Proceedings of the 41st Hawaii International Conference on System Sciences*, page 106, Big Island, Hawaii, January 2008. IEEE Computer Society Press.
- [6] JOE GREGORIO and BILL DE HÓRA. The Atom Publishing Protocol. Internet RFC 5023, October 2007.
- [7] IAN JACOBS and NORMAN WALSH. Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215, December 2004.
- [8] MARIO JECKLE and ERIK WILDE. Identical Principles, Higher Layers: Modeling Web Services as Protocol Stack. In *Proceedings of XML Europe 2004*, Amsterdam, Netherlands, April 2004.
- [9] ANDREW P. MCAFEE. Enterprise 2.0: The Dawn of Emergent Collaboration. *MIT Sloan Management Review*, 47(3):21–28, 2006.
- [10] MARK NOTTINGHAM and ROBERT SAYRE. The Atom Syndication Format. Internet RFC 4287, December 2005.
- [11] MARK NOTTINGHAM. FIQL: The Feed Item Query Language. Internet Draft draft-nottingham-atompub-fiql-00, December 2007.
- [12] J. DOUGLAS ORTON and KARL E. WEICK. Loosely Coupled Systems: A Reconceptualization. *Academy of Management Review*, 15(2):203–223, April 1990.
- [13] BIJAN PARSIA and PETER F. PATEL-SCHNEIDER. OWL 2 Web Ontology Language: Primer. World Wide Web Consortium, Working Draft WD-owl2-primer-20080411, April 2008.
- [14] CESARE PAUTASSO, OLAF ZIMMERMANN, and FRANK LEYMANN. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In *Proceedings of the 17th International World Wide Web Conference*, pages 805–814, Beijing, China, April 2008. ACM Press.
- [15] CYNTHIA RETTIG. The Trouble with Enterprise Software. *MIT Sloan Management Review*, 49(1):21–27, 2007.
- [16] JIM SPOHRER, PAUL P. MAGLIO, JOHN BAILEY, and DANIEL GRUHL. Steps Toward a Science of Service Systems. *IEEE Computer*, 40(1):71–77, January 2007.
- [17] DON TAPSCOTT and ANTHONY D. WILLIAMS. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, New York, NY, December 2006.
- [18] STEVE VINOSKI. Serendipitous Reuse. *IEEE Internet Computing*, 12(1):84–87, January

- 2008.
- [19] ERIK WILDE. The Plain Web. In *Proceedings of the First International Workshop on Understanding Web Evolution (WebEvolve2008)*, pages 79–83, Beijing, China, April 2008.