In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective

Dominique Guinard, Iulia Ion, and Simon Mayer

Institute for Pervasive Computing, ETH Zurich, Switzerland,
dguinard|iulia.ion|simon.mayer@inf.ethz.ch

Abstract. Current trends inspired from the development of the Web 2.0 advocate designing smart things (e.g., wireless sensors nodes or home appliances) as service platforms. Interoperable services are mainly achieved using two different approaches: WS-* and RESTful Web services. These approaches have previously been compared with respect to performance and features, but no work has been done to elicit the developers' preferences and programming experiences. We conducted a study in which 69 novice developers learned both technologies and implemented mobile phone applications that retrieve sensor data, both through a RESTful and through a WS-* service architecture. The results complement the available technological decision framework when building Internet of Things applications. The results suggest that developers find REST easier to learn than WS-* and consider it more suitable for programming smart things. However, for applications with advanced security and Quality of Service requirements, WS-* Web services are perceived to be better suited.

Key words: Internet of Things, Web of Things, WSN, Mobile, Web Services, REST, WS-*

1 Introduction and Related Work

The *Internet of Things* (IoT) explores ways to connect and build upon networks of digitally enhanced, communication-capable objects, such as wireless sensor nodes, mobile phones, and home appliances, generally known as "smart things". Sensor nodes might, for instance, be networked together to create environmental monitoring applications [13].

Home appliances might communicate with each other to offer smarter heating systems or to optimize their energy consumption [5]. However, the development of applications that integrate the functionality of multiple smart things remains a challenging task, because it requires expert knowledge of each platform.

To facilitate this, recent research initiatives tried to provide uniform interfaces that create a loosely coupled ecosystem of services for smart things [5, 10]. The goal is to enable a widely distributed platform in which smart things provide services that can be easily composed to create new applications. Two types

of service-oriented architectures stand out as potential candidates to enable uniform interfaces to smart objects: the Representational State Transfer (REST) [3] and WS-* [9] Web services:

*WS-** These services declare their functionality and interfaces in a Web Services Description Language (WSDL) file. Client requests and service response objects are encapsulated using the Simple Object Access Protocol (SOAP) and transmitted over the network, usually using the HTTP protocol. Further WS-* standards define concepts such as addressing, security, discovery or service composition. Although WS-* was initially created to achieve interoperability of enterprise applications, work has been done to adapt it to the needs of resource-constrained devices [10, 13]. Furthermore, lighter forms of WS-* services, such as the Devices Profile for Web Services (DPWS)[1], were proposed [6].

REST At the core of a RESTful architecture [3] lie *resources* that are uniquely identified through Uniform Resource Identifiers (URIs). The Web is an implementation of RESTful principles – it uses URLs to identify resources and HTTP as their service interface. Resources can have several representation formats (e.g., HTML, JSON[2]) negotiated at run time using HTTP content negotiation. In a typical REST request, the client discovers the URL of a service it wants to call by browsing or crawling its HTML representation. The client then sends an HTTP call to this URL with a given verb (`GET`, `POST`, `PUT`, etc.), a number of options (e.g., accepted format), and a payload in the negotiated format (e.g., XML or JSON). Several recent research projects implement RESTful Web services for smart things [2] within what has become to be known as the *Web of Things* [5].

While the two architectures share the goal of providing developers with abstractions (i.e., APIs) for interacting with distributed services, they tackle loose coupling and interoperability differently. Consequently, work has been done to evaluate the two approaches. In [8, 9], REST and WS-* are compared in terms of re-usability and loose coupling for business applications. The authors suggest that WS-* services should be preferred for "professional enterprise application integration scenarios" and RESTful services for tactical, ad-hoc integration over the Web.

Internet of Things applications pose novel requirements and challenges as neither WS-* nor RESTful Web services were primarily designed to run and be used on smart things, but rather on business or Web servers. This development thus necessitates assessing the suitability of the two approaches for devices with limited capabilities. Yazar et al. [13] analyze the performance of WS-* and RESTful applications when deployed on wireless sensor nodes with limited resources and conclude that REST performs better.

However, evaluating the performance of a system when deployed on a particular platform is not enough to make the architectural decision that will foster adoption and third-party (public) innovation. Indeed, studies like the Technology Acceptance Model [1] and more recent extensions [4] show that the perceived

[1] See `www.ws4d.org`
[2] See `www.json.org`

ease of use of an IT system is key to its adoption. As many manufacturers of smart things are moving from providing devices with few applications to building devices as platforms with APIs, they increasingly rely on external communities of developers to build innovative services for their hardware (e.g., the Apple App Store or Android Marketplace). An easy to learn API is, therefore, key in fostering a broad community of developers for smart things. Hence, choosing the service architecture that provides the best developer experience is instrumental to the success of the Internet of Things and the Web of Things on a larger scale.

In this paper we complement the decision framework that can be used when picking the right architecture for IoT applications and platforms. We supplement previous work [8, 9, 13] by evaluating, in a structured way, the actual *developers' experience* when using each architecture in an IoT context. We analyze the perceived ease of use and suitability of WS-* and RESTful Web service architectures for IoT applications. Our study is based on the qualitative feedback and quantitative results from 69 computer science students who developed two applications that accesses temperature and light measurements from wireless sensor nodes. For one of the applications, the participants used a sensor node offering a RESTful Web API. In the second case, they were accessing a sensor node through a WS-* (WSDL + SOAP-based) API.

Our results show that participants almost unanimously found RESTful Web services easier to learn, more intuitive and more suitable for programming IoT applications than WS-*. The main advantages of REST as reported by the participants are *intuitiveness*, *flexibility*, and the fact that it is more *lightweight*. WS-* is perceived to support more advanced *security* requirements and benefits from a clearer *standardization* process.

This paper is structured as follows. Section 2 describes the study methodology. Section 3 presents and analyses the results. Finally, Section 4 discusses the implications of our findings and devises guidelines.

2 Methodology

Our study is based on a programming exercise and the feedback received from a group of 69 computer science students who learned about RESTful and WS-* Web service architectures and implemented, in teams, mobile phone applications that accessed sensor data from different sensor nodes using both approaches. The exercise was part of an assignment in the Distributed Systems course at ETH Zurich[3]. The material used for instructing the students about the two technologies was prepared so as to not introduce a bias[4].

Although we worked on native architectures for both REST and WS-* [5], in order for the results not to be influenced by the performance of the sensors for each architecture, we used two proxies as shown in Figure 1. As a result, the

[3] The assignment is available online: `tinyurl.com/vs-assignment`
[4] All course material is available online: `tinyurl.com/vs-material`

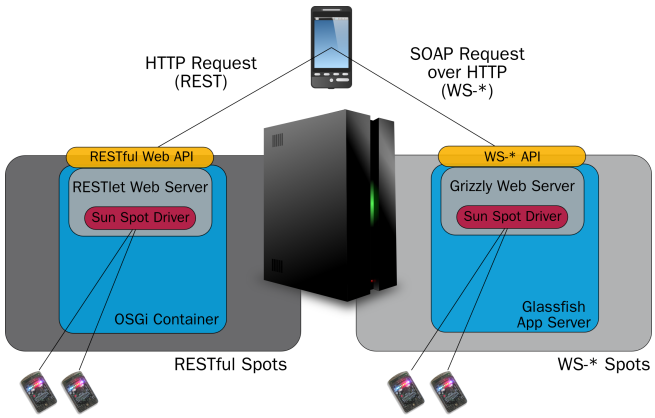

Fig. 1. Setup of the user study. In order not to influence the results, the 4 Sun SPOTs sensor nodes are connected through proxies offering once a RESTful API (left), once a WS-* API (right).

study can focus on the development experience and usability rather than on the architectures' performance that has already been studied by others (e.g., [13]).

To get and parse the RESTful sensor responses, participants were advised to use the Apache HTTP Client library[5] and JSON.org libraries[6]. To perform the WS-* request, we advised the students to use the *kSoap2* library[7] which offers a set of tools to interact with WS-* services. It is worth noting that while these two libraries are well aligned with the abstractions represented by both architectures and are standard tools for accessing each type of API from a mobile device, the students were free to pick other tools. Averaging over the submissions, the programs had 105 lines of code for the WS-* implementation ($SD = 50.19$, where SD is the Standard Deviation), opposed to 98 lines of code for the REST implementation ($SD = 48.31$).

The two coding tasks were solved in teams of two or three members who were able to freely decide how to split up the tasks amongst team members. The coding tasks were successfully completed by all teams within two weeks. To ensure that every team member had understood the solution of each task, individual structured questionnaires about both technologies had to be submitted. Additionally, we included a voluntary feedback form on the learning process in the study. Students were informed that answers in the feedback form were not part of the assignment, and that responses would be used in a research study. To encourage them to give honest answers about the amount of effort invested in solving both coding tasks, perception, and attitudes towards both technologies, entries in the feedback form were made anonymously. Table 1 summarizes the data collection sources.

[5] See `hc.apache.org`
[6] See `json.org`
[7] See `ksoap2.sourceforge.net`

Demographics: The participants were from ETH Zurich, in their third or fourth year of Bachelor studies. Teams were formed of two or three members. They were taught both technologies for the first time, in a short introduction during the tutorial class. 89% reported not having had any previous knowledge of WS-* and 62% none of REST. From the 35% that had already used REST before the course, half reported that they had previously been unaware that the technology they used actually was based on the REST architecture. 5% (2 students) had programmed WS-* applications.

Additional Tasks: Subsequent tasks involved creating visualization mechanisms for the retrieved data, both locally and through cloud visualization solutions. These tasks are, however, not relevant for the presented study.

Data Source	**Type**	**N**
RESTful and WS-* Applications	Team	25
Structured Questionnaire	Individual	69
Voluntary Feedback Form	Anonymous	37

Table 1. Data was collected from different programming tasks and questionnaires.

3 Results

In this section, we present our results on the perceived differences, ease of learning, and suitability of the technologies for IoT-related use cases.

3.1 Perceived Differences

Using the structured questionnaire, we collected qualitative data on the perceived advantages of both technologies with respect to each other. While REST was perceived to be "*very easy to understand, learn, and implement,*" lightweight and scalable, WS-* "*allows for more complex operations,*" provides higher security, and a better level of abstraction. Table 2 summarizes the perceived advantages of the two technologies.

3.2 Accessibility and Ease of Learning

In the feedback form, we asked participants to rate on a 5 point Likert scale how easy and how fast it was to learn each technology (1=not easy at all, ..., 5=very easy). As shown in Figure 3, 70% rated REST "easy" or "very easy" to learn. WS-* services, on the other hand, were perceived to be more complex: only 11% respondents rated them easy to learn. Event if all participants were required to learn and explain the concepts of both technologies, compare their advantages, analyze their suitability and explain design decisions, we restricted

REST ($N = 69$)	#
Easy to understand, learn, and implement	36
Lightweight	27
Easy to use for clients	25
More scalable	21
No libraries required	17
Accessible in browser and bookmarkable	14
Reuses HTTP functionality (e.g., caching)	10
WS-* ($N = 69$)	#
WSDL allows to publish a WS-* interface	31
Allows for more complex operations	24
Offers better security	19
Provides higher level of abstraction	11
Has more features	10

Table 2. Participants felt that WS-* provides more features, but REST is easy to learn and use.

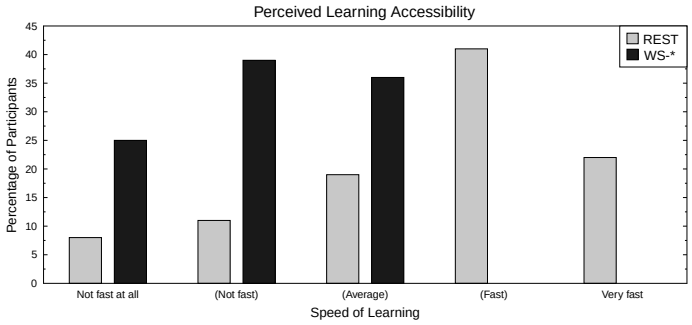

Fig. 2. A majority of participants reported that REST as *fast* or *very fast* to learn and WS-* as *not fast* or *average*.

the sample to participants who reported to have worked on programming both REST and WS-* assignments within their teams (N=19) to avoid bias. We then applied the Wilcoxon signed rank test for the two paired samples to compare the perceived ease of learning for REST and WS-*. Our results show that REST (with an average $M = 3.85$ and a Standard Deviation $SD = 1.09$) was reported to be statistically significantly easier to learn than WS-* ($M = 2.50, SD = 1.10$): $V = 153, p < 0.001$. Similarly, REST ($M = 3.43, SD = 1.09$) was perceived to be significantly faster to learn than WS-* ($M = 2.21, SD = 0.80$), $V = 53, p < 0.009, N = 14$.

Furthermore, in the feedback form, we collected qualitative data on the challenges of learning both technologies, asking the participants: "What were the challenges you encountered in learning each of the technologies? Which one was faster and easier to learn?". Nine participants explained that REST was easier

and faster to learn because RESTful Web services are based on technologies, such as HTTP and HTML, which are well-known to most tech-savvy people: "*Everybody who is using a browser already knows a little about [REST].*"

WS-* was perceived to be overly complicated: "*REST is easy and WS-* is just a complicated mess.*" Reasons for such strong statements were the complexity of extracting useful information out of the WSDL and SOAP files (mentioned by 8), as well as the little and poor documentation of parameters for a SOAP call. The lack of clear documentation was perceived as a problem for REST as well: Seven participants said that further request examples, alongside with the traditional documentation (e.g., Javadoc) for both REST and WS-*, would be very useful. Eight participants explicitly mentioned that they had had previous experience with REST during their spare time. This illustrates the accessibility and appeal of RESTful Web services, and it positions them as an ideal candidate for smart things APIs in terms of lowering the entry barrier for creating applications. In the feedback form, 25 participants said that REST made it easier to understand what services the sensor nodes offered. Eight participants explained this by the fact that, for REST, an HTML interface was provided. This emphasizes that RESTful smart things should offer an HTML representation by default. Seven participants found WS-* easier for this matter. They noted that a WSDL file was not necessarily easy to read, but they liked the fact that it was "*standard*".

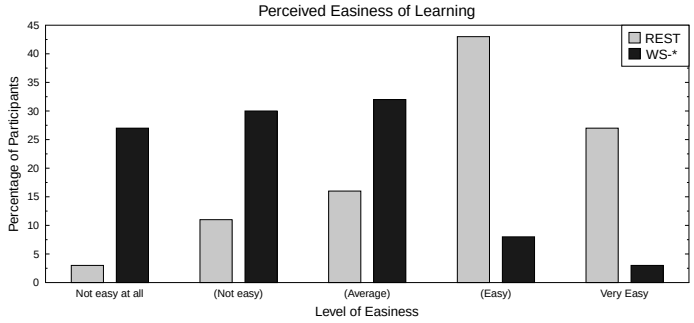

Fig. 3. Participants reported REST as easier to learn than WS-*.

3.3 Suitability for Use-Cases

In the feedback form, we asked participants to rate on a Likert scale (1=WS-*, ..., 5=REST) which one of the two technologies they would recommend in specific scenarios. REST was considered more suitable than WS-* for IoT applications running on embedded devices and mobile phones (see Figure 4). The one sample Wilcoxon signed rank test confirmed that the sample average was statistically

higher than the neutral 3, and therefore inclined towards REST. This was the case both for embedded devices ($M = 3.86, SD = 1.03, V = 342, p < 0.001$), and for mobile phone applications ($M = 3.51, SD = 1.12, V = 252, p < 0.007$). For business applications, however, a higher preference for WS-* was stated but not statistically significant ($M = 2.67, SD = 1.33, V = 163.5, p = 0.12$).

General Use-Cases We asked our participants to discuss the general use-cases for which each technology appeared suitable. When asked: "For what kind of applications is REST suitable?", 23 people mentioned that REST was well adapted for simple applications offering limited and atomic functionality: "*for applications where you only need create/read/update and delete [operations]*". 8 participants also advised the use of REST when security is not a core requirement of the application: "*Applications where no higher security level than the one of HTTP[s] is needed*". This is supported by the fact that the WS-* security specification offers more levels of security than the use of HTTPS and SSL in REST [11]. 6 participants suggested that REST was more adapted for user-targeted applications: "*[...] for applications that present received content directly to the user*". Along these lines, 14 users said that REST was more adapted for Web applications or applications requiring to integrate Web content: "*[for] Web Mashups, REST services compose easily*".

We then asked: "For what kind of applications is WS-* more suitable?". 20 participants mentioned that WS-* was more adapted for secure applications: "*applications that require extended security features, where SSL is not enough*". 16 participants suggested to use WS-* when strong contracts on the message formats were required, often referring to the use of WSDL files: "*with WS-* [...] specifications can easily be written in a standard machine-readable format (WSDL, XSD)*".

REST (N=37)	#
For simple applications, with atomic functionality	23
For Web applications and Mashups	14
If security is not a core requirement	8
For user-centered applications	6
For heterogeneous environments	6
WS-* (N=37)	#
For secure applications	20
When contracts on message formats are needed	16

Table 3. REST was perceived to be more suited for simple applications, and WS-* for applications where security is important.

For Smart Things Both WS-* and RESTful Web Services were not primarily designed to run on embedded devices and mobile phones but rather on business or Web servers. Thus, assessing the suitability of the two approaches for devices with limited capabilities is relevant.

As shown in the first part of Figure 4, for providing services on embedded devices, 66% of the participants suggested that REST was either "adapted" or "very-adapted". When asked to elaborate on their answers, 6 participants suggested that for heterogeneous environments, REST was more suitable: "*for simple application, running on several clients (PC, iPhone, Android) [...]*". 7 participants said that REST was adapted for embedded and mobile devices because it was more lightweight and better suited for such devices in general: "*the mapping of a sensor network to the REST verbs is natural [...]*". To confirm this, we investigated the size of the application packages for both approaches. The average footprint of the REST application was 17.46 kB while the WS-* application had a size of 83.27 kB on average. The difference here is mainly due to the necessity to include the *kSoap2* library with every WS-* application. These results confirm earlier performance and footprint evaluations [2, 5, 13].

For Smart Home Applications We then went into more specific use cases, asking: "*Imagine you want to deploy a sensor network in your home. Which technology would you use and why?*". Sixty-two respondents recommended REST to deploy a sensor network in the home, 5 recommended WS-*, and 2 were undecided. Twenty-four participants justified the choice of REST by invoking its simplicity both in terms of use and development: "*REST [...] requires less effort to be set up*", "*Easier to use REST, especially in connection with a Web interface*". Eight participants said that REST is more lightweight, which is important in a home environment populated by heterogeneous home appliances. Interestingly, 14 participants mentioned that in home environments there are very little concerns about security and thus, the advanced security features of WS-* were not required: "*I would not care if my neighbor can read these values*", "*The information isn't really sensitive*".

For Mobile Phones Since the mobile phone is a key interaction device for creating IoT applications, we asked the participants to assess the suitability of each platform for creating mobile phone clients to smart things. As shown in the second part of Figure 4, 53% of the participants would use REST, 16% would use WS-* and 32% were undecided. They explained these contrasted results by the fact that mobile phones are getting very powerful. 7 participants explained that the amount of data to be processed was smaller with REST which was perceived as an important fact for mobile clients. Interestingly, some participants considered the customers of mobile platforms to have different requirements: "*I would use REST, since customers prefer speed and fun over security for smaller devices*". The lack of native WS-* support on Android (which natively supports HTTP) and the required use of external libraries was also mentioned as a decision factor as REST calls can be simply implemented only using the native HTTP libraries.

For Business Applications The results are much more inclined towards WS-* when considering "business" applications. As shown in the third part of Figure 4, the majority of our participants (52%) would choose WS-* and 24% REST for servicing business applications. Twenty-one (out of 69, see Table 1) justify

their decision by the security needs of enterprise applications: "*I would rely on the more secure WS-* technology*". Eighteen participants talk about the better described service contracts when using WS-*: "*I propose WS-* because we could use WSDL and XSD to agree on a well-specified interface early on [...]*". Amongst the participants suggesting the use of REST, 10 justify their decision with its simplicity and 10 with its better scalability.

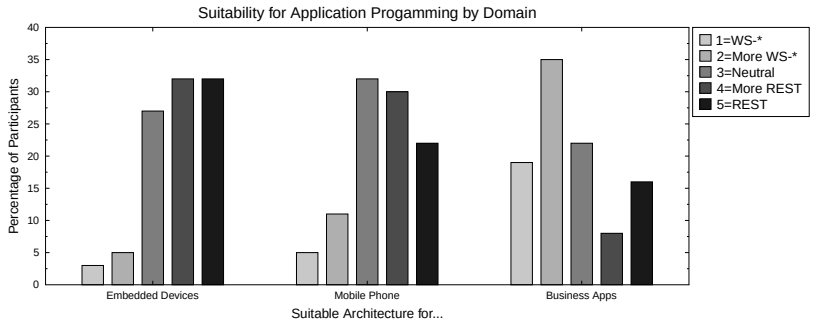

Fig. 4. Participants reported that REST is better suited for Internet of Things applications, involving mobile and embedded devices and WS-* fits better to the requirements of business applications (N = 69).

4 Discussion and Summary

A central concern in the Internet of Things and thus in the Web of Things is the interoperability between smart objects and existing standards and applications. Two service-oriented approaches are currently at the center of research and development: REST and WS-*. Decisions on which approach to adopt have important consequences for an IoT system and should be made carefully. Our contribution is to complement existing studies on performance metrics with an evaluation of the developers' preferences and IoT programming experiences with REST and WS-*. Our results show that, in the context of the conducted study, REST stands out as the favorite service architecture. We summarize the decision criteria used by developers in our study and devise guidelines in Table 4.

Future studies should conduct a long-term assessment of the developers' experience, beyond the initial phase of getting started with the technologies. As an example it would be interesting to address the clear trend (52% for WS-*, 24% for REST) towards choosing a WS-* architecture when considering business applications even if overall the participants favored REST and mentioned its superiority for several important factors in business environments such as scalability or interoperability. Are their decisions solely based on the superior security features of WS-* or is there a perception bias?

Furthermore, some of the participants' remarks (e.g., on the low security requirements in home environments) should be put in the context of relatively

novice developers. Hence, future work could be done to compare the experience of advanced developers, possibly within industry projects. However, the more experienced the developers are, the more they are likely to develop a bias towards one or the other technology.

Requirement	REST	WS-*	Justification
Mobile & Embedded	+	-	Lightweight, IP/HTTP support
Ease of use	++	-	Easy to learn
Foster third-party adoption	++	-	Easy to prototype
Scalability	++	+	Web mechanisms
Web integration	+++	+	Web is RESTful
Business	+	++	QoS & security
Service contracts	+	++	WSDL
Adv. security	-	+++	WS-Security

Table 4. Guidelines for choosing a service architecture for IoT platforms.

While our results confirm several other research projects that take a more performance-centric approach [2, 5, 13], they contradict several industry trends. In home and industrial automation, standards such as UPnP, DLNA or DPWS expose their services using WS-* standards. One of the reasons for this, also noted by participants, is the lack of formal service contracts (such as WSDL and SOAP) for RESTful services. This is an arguable point as Web experts [11] already illustrated how a well-designed RESTful interface combined with HTTP content negotiation results in a service contract similar to what WS-* offers [11], with no overhead and more adapted to services of smart things [5]. Yet, this illustrates an important weakness of RESTful Web services: RESTful Web services are a relatively *fuzzy* concept. Even if the basics of REST are very simple, the lack of a clear stakeholder managing "standard" RESTful architectures is subject to many (wrong) interpretations of the concept. Until this is improved, resources such as [3, 11] profile themselves as de facto standards.

In cases with strong security requirements, WS-* has a competitive advantage [9, 11]. The WS-Security standard offers a greater number of options than HTTPS (TLS/SSL) such as encryption and authentication beyond the communication channel, endpoint-to-endpoint. In theory, these could also be implemented for RESTful Web services. However, the lack of standard HTTP support of these techniques would result in tight coupling between the secured things and their clients. Nevertheless, in the context of smart things, it is also important to realize that WS-Security standards are much more resource intensive than those of HTTPS and, thus, rarely fit resource-constrained devices.

Finally, it is important to consider *how accessible smart things should be*. Participants identified that RESTful Web services represent the most straightforward and simple way of achieving a global network of smart things because RESTful Web services seamlessly integrate with the Web. This goes along the lines of recent developments, such as 6LoWPAN [7] and the IPSO alliance[8],

[8] See `ipso-alliance.org`

CoRE[9] and CoAP [12], or the Web of Things Architecture [5], where smart things are increasingly becoming part of the Internet and the Web.

References

1. Fred D. Davis. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3):319–340, 1989.
2. W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin. pREST: a REST-based protocol for pervasive systems. In *Proc. of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 340–348. IEEE, 2004.
3. R. Fielding. *Architectural styles and the design of network-based software architectures*. Phd thesis, 2000.
4. David Gefen and Mark Keil. The impact of developer responsiveness on perceptions of usefulness and ease of use: an extension of the technology acceptance model. *SIGMIS Database*, 29:35–49, April 1998.
5. Dominique Guinard, Vlad Trifa, and Erik Wilde. A Resource Oriented Architecture for the Web of Things. In *Proc. of the 2nd International Conference on the Internet of Things (IoT 2010)*, LNCS, Tokyo, Japan, November 2010. Springer Berlin / Heidelberg.
6. F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62–70, 2005.
7. Geoff Mulligan. The 6LoWPAN architecture. In *Proc. of the 4th workshop on Embedded networked sensors (EmNets '07)*, EmNets '07, pages 78–82, Cork, Ireland, 2007. ACM.
8. Cesare Pautasso and Erik Wilde. Why is the web loosely coupled?: a multi-faceted metric for service design. In *Proc. of the 18th international conference on World Wide Web (WWW '09)*, pages 911–920, Madrid, Spain, April 2009. ACM.
9. Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big web services: making the right architectural decision. In *Proc. of the 17th international conference on World Wide Web (WWW '08)*, pages 805–814, New York, NY, USA, 2008. ACM.
10. N.B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *Proc. of the 6th ACM conference on Embedded Network Sensor Systems (SenSys '08)*, pages 253–266, Raleigh, NC, USA, 2008. ACM.
11. Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, May 2007.
12. Z. Shelby. Embedded web services. *IEEE Wireless Communications*, 17(6):52–57, December 2010.
13. Dogan Yazar and Adam Dunkels. Efficient application integration in IP-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, page 4348, Berkeley, CA, USA, November 2009.

[9] See `tools.ietf.org/wg/core`