Towards Physical Mashups in the Web of Things

Dominique Guinard, Vlad Trifa
SAP Research Zurich and
Institute for Pervasive Computing ETH Zurich, Switzerland
Contact email: dguinard@ethz.ch

Thomas Pham, Olivier Liechti
University of Applied Sciences HEIG-VD
Institute for Information Technologies, Switzerland

***Abstract*—Wireless Sensor Networks (WSNs) have promising industrial applications, since they reduce the gap between traditional enterprise systems and the real world. However, every particular application requires complex integration work, and therefore technical expertise, effort and time which prevents users from creating small tactical, ad-hoc applications using sensor networks. Following the success of Web 2.0 "mashups", we propose a similar lightweight approach for combining enterprise services (e.g. ERPs) with WSNs. Specifically, we discuss the traditional integration solutions, propose and implement an alternative architecture where sensor nodes are accessible according to the REST principles. With this approach, the nodes become part of a "Web of Things" and interacting with them as well as composing their services with existing ones, becomes almost as easy as browsing the web.**

I. Introduction and Related Work

In order to be globally competitive, enterprises need efficient IT systems that provide comprehensive and timely information. This implies the need for a continuous information flow, from industrial machines up to business applications, and therefore the integration of many heterogeneous systems. Nowadays, enterprise application integration has already gone beyond the interconnection of a few large back-end systems. However, this integration does not stop here: more and more everyday objects are now augmented with computing and communication capabilities. With technologies like WSNs and RFID, the physical world becomes thus "integrable" with computer networks, and makes the state of objects and their surroundings seamlessly accessible to software systems. The "Internet of Things" [1] is a global communication network that is emerging with the dissemination of such devices, and is rapidly expanding.

As a matter of fact, most projects in this field are based on monolithic system architectures, which are brittle and difficult to adapt. While some general purpose frameworks exist, developing a new application in a specific domain requires strong skills across a wide spectrum of technologies. For this reason, smart devices and WSNs are still underexploited. In the area of enterprise application integration, a few projects have explored the use of "mashup" architectures, also known as user-generated composite applications, to enable more flexible software composition within (and outside) the enterprise [2], [3]. However, they mainly focus on mashing up on-line services and do not address the issues and requirements that come with a physical world integration, which is our aim. Some issues of integration with the physical world are discussed in [4], [5]. In these approaches, UPnP or the Web Services standards (WS-*) are used for integration. The complexity of these underlying protocols implies a steep learning curve and is geared towards software engineers. We propose exploring a different approach that empowers individual end-users with moderate computer literacy to create simple, ad-hoc applications that combine real-time data and services provided by sensor nodes with enterprise services: the "physical mashups".

Dickerson et *et al.* [6] have used Web feeds to access data provided by sensor nodes. In particular, they describe an extension to RSS better suited to accommodate high-rate data streams with a web-oriented querying interface to retrieve sensor data. A direct consequence of the stream abstraction is that sensors are considered solely as data publishers, not as service providers. In [7] a RESTful architecture for sensor networks is proposed and evaluated. Similarly to [8] we push the REST principles (see [9] and subsection II-B) further down to the sensor level rather than the gateway level, which goes along with the developments of IP stacks for WSNs. In order to further diminish the footprint of the application on the nodes, we propose the use of JSON[1] as a data interchange format. JSON is a lightweight alternative to the XML format used in [7], [8]. Our main contribution is on the application level. Following on the web paradigm we expose the functionalities of sensor nodes as web resources and link them together hierarchically, like web pages. This reduces searching for services to web-browsing. We then discuss how this architecture can facilitate the creation of ad-hoc applications by end-users and discuss the alternative architectures. As an example we consider a composite application, that could be built by an end-user, where a sensor node updates the temperature status of a shipment in an ERP (Enterprise Resource Planning) application on a regular basis.

II. Approaches for System Integration

Just as "digital mashups" are getting useful for business [3] and everyday applications, we believe that "physical mashups" will become so. Still, the current software integration platforms are not adapted to ad-hoc scenarios, because they lack simple and open interfaces for simple purposes. To illustrate this we start by discussing the common integration techniques used to link the physical world and enterprise applications.

[1] http://www.json.org

A. Custom Middleware Integration

In the world of business software, specialized enterprise middleware is often used to create an abstraction layer between the manufacturing plant devices with management applications. In the field of manufacturing, such a middleware is called a Manufacturing Execution System (MES). The problem is that today, most devices expose proprietary interfaces and communication protocols. For this reason, their integration into an enterprise system requires a substantial effort. For every type of device, a custom adapter must be developed and integrated into the enterprise middleware. This approach may be sufficient for rather static environments where the set of device types is known and does not evolve often. However, it is not suitable for dynamic environments where new kinds of devices and machines need to be integrated on a regular basis. In addition, middleware solutions are often fairly complex systems, which makes them hard to use them for ad-hoc and simple service composition.

B. Classic Web Service Oriented Approach

To overcome the burden of proprietary interfaces, WS-* Web Services standards (also known as WS-* stack, or Big Web Services [10]) have been proposed. Web Services are loosely coupled software components that offer standardized interfaces based on mainly two languages: the Web Service Definition Language (WSDL) which is used to define the syntax of the interfaces, and SOAP which defines the format of messages that are exchanged when invoking services. The WS-* approach was initially designed to propose a distributed component architecture that could be used to integrate and combine different computer systems, with a clear emphasis on business architectures and for digital services (but not on the physical world). This emphasis resulted in relatively heavy standards: both the interface definition (WSDL) and the messages (SOAP) are rather complicated instances of XML documents. This makes the WS-* services quite demanding in terms of required computing power, bandwidth and storage.

Several initiatives attempted to apply the WS-* paradigm beyond the closed-world of servers and mainframes. For example, in SOCRADES [5] Web service standards are pushed down to the device level, so that all the actors of the physical world (sensors, machines, augmented objects, etc.) can expose their services through a common interfacing language. The Device Profile for Web Services (DPWS, a subset of WS-* standards) can be used to enable Web service messaging, discovery, description and eventing on devices with constrained resources [11]. However, DPWS needs to be further developed in order to fit the needs of WSNs as its current footprint is still too big for most nodes. While this problem is probably a matter of time, it remains that DPWS uses a complex set of standards. Based on our experience within SOCRADES [5] we can state that it requires significant expert knowledge and tools that average users do not posses. We suggest that this stack is suited for well-defined integration scenarios, but remains too complicated for ad-hoc integration scenarios by end-users.

C. Resource Oriented Approach: RESTful Web Services

The Internet is a stunning example of a scalable global network of computers that interoperate smoothly across heterogeneous hardware and software platforms. It also illustrates well how simple and open standards (e.g., HTTP, XML) can be used to build very efficient and flexible systems. The architectural principle that is at the heart of the Web, namely Representational State Transfer (REST) as defined by Roy Fielding [9], shares a similar goal with the WS-* standards: increase interoperability for a looser coupling between parts of distributed applications. However, the goal of REST is to achieve this in a more lightweight and simpler manner, by reusing web patterns. In particular, REST uses the Web as an application platform and fully leverages all the features inherent to HTTP such as authentication, authorization, encryption, compression and caching. Moreover, it brings services "into the browser": the resources can be linked and bookmarked and the results are directly visible in any Web browser, without the need for generating complicated source code out of WSDL files for interacting with the service. These advantages mainly explain why REST services are the technological basis for an increasing number of Web 2.0 services such as Flickr, Facebook, Del.icio.us, Doodle, or Amazon.

To achieve this, REST proposes two basic rules: 1) The application model is transformed from operation-centric (i.e. service operations in WS-*) into a data-centric one. This means "everything" that offers services becomes a resource (e.g. a temperature sensor is a resource of a the sensor node resource) that can be identified unambiguously using a URI. 2) The four main operations provided by HTTP (GET, POST, PUT, DELETE) are the only available operations on resources, they define a uniform interface.

REST services also have certain limitations, however. For instance, the inherent simplicity of REST paradoxically complicates the creation of complex services. But Pautasso *et al.* [12] compared WS-* services with REST services and suggested that WS-* services are to be preferred for "professional enterprise application integration scenarios, with a longer lifespan and advanced QoS requirements". Conversely, thank to their simplicity, lightweighness and uniform interface, RESTful service are preferred for "tactical, ad-hoc integration over the Web".

III. A REST Architecture for Wireless Sensor Networks

In order to empirically test the potential advantages of the Web and HTTP principles, we implemented a RESTful architecture on the SunSPOT Java sensor nodes. The architecture is composed of three subparts implemented in Java Micro Edition (CLDC): the software deployed on the sensor nodes themselves, the software deployed on the gateway, and the enterprise integration software.

A. Embedded Software Stack

Integration at a lower-level is also facilitated by using RESTful interactions with devices. For that, we implemented

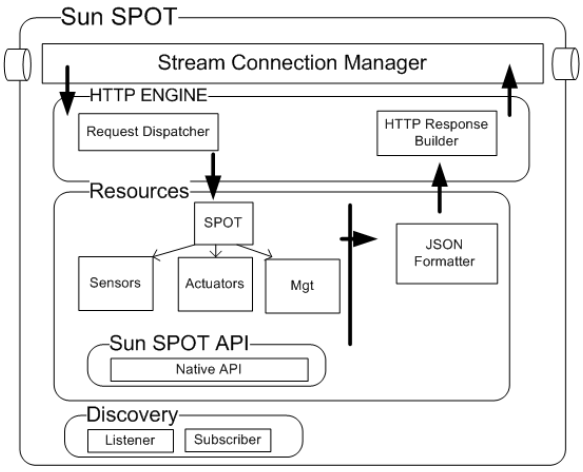

Fig. 1. Architecture deployed on the SunSPOTS.

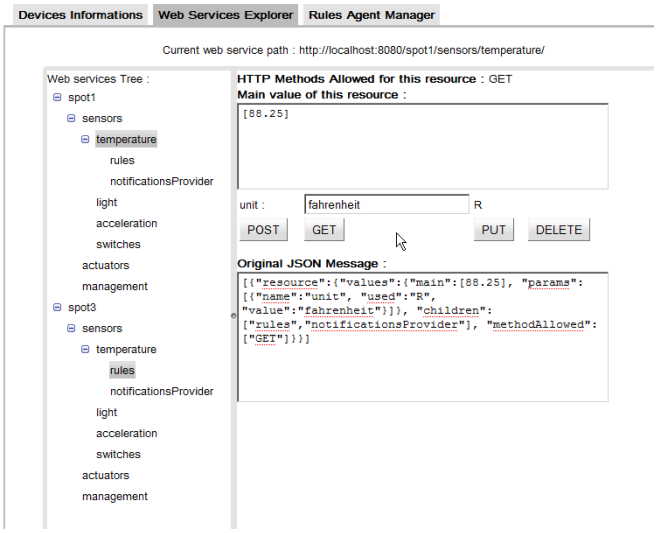

Fig. 2. Using the AJAX Resource Explorer, users can explore directly the resources provided by the device. This sample shows how users can get the current temperature resource values and the formalism of the JSON response.

an embedded HTTP server directly on the SunSPOT nodes that natively supports the four main operations of the HTTP protocol (GET, POST, PUT, DELETE, i.e the *verbs* of REST). The HTTP server is deployed on each sensor node, making it an independent and autonomous device. Each SunSPOT offers a number of sensors (light, temperature, accelerometer, etc.), a number of actuators (digital outputs, leds, etc.) and a number of internal components (radio, battery). These (including the SunSPOT themselves) are the *resources* of our REST architecture. Resources are organized in a tree hierarchy and each of them implements or inherits the four verbs.

Requests for services (i.e. verbs on resources) are formulated using a standard URL. For instance, typing a URL such as http://webofthings.com:8080/spot1/sensors/temperature in a browser, requests the resource "temperature" of the resource "sensor" of "spot1" with the verb GET. The request is routed by the *RequestDispatcher* to the correct resource (see Fig. 1) on which it invokes the `doGet()` operation. The resource then reads the current temperature using the native SunSPOT API and sends it to the formatter. While this component can support various format, we decided to use JSON (JavaScript Object Notation), an alternative to XML often used as an data-interchange format for web mashups. Since JSON is a lightweight format we believe it is more adapted to devices with limited capabilities both because the amount of data transfered is reduced and the parsers require less resources. The JSON data resulting from the call for temperature is shown on Figure 2. This data is finally wrapped into an HTTP packet and sent further to the gateway. Note that alternatively the results can be distributed asynchronously to a URL when the values overcome a certain threshold configurable through the REST interface as well.

B. Gateway Middleware

SunSPOTs do not support the IP (Internet Protocol) stack (yet?). Their radio communication is based on the IEEE 802.15.4 standard. The Web is not directly linked to this protocol, thus a gateway that bridges the Web requests (from TCP/IP) and to the devices over the IEEE 802.15.4 link is needed. To allow flexible mashups, we wanted the nodes to be mobile, travelling from gateways to gateways, which requires a dynamic discovery process to find new nodes and registers their basic information (the MAC address, a short description, their URL). This process is carried out by a *Discovery Component*, which broadcasts invitation messages on a regular basis on a dedicated port. On their side, the nodes listen to this port and can decide to subscribe to the broadcasting gateway. Then *Reverse Proxy* registers the node's address. When receiving an HTTP request from the Internet/Intranet it reads the request URL and maps it to one of the registered nodes. In case the node is busy, it also serves as a buffer by queuing requests and resubmitting them later. In order to deal with URL and HTTP packets parsing, the gateway uses and extends the functionalities of RESTlet, an open-source Java REST library.

As shown on Figure 2, the *Resource Explorer* component offers to users (i.e. mashup builders) an interface to browse the available services. Just as they would navigate on websites, they can explore the device hierarchy and test services by clicking on the link structure reflecting the hierarchy of the physical world (e.g. a temperature sensor is the "son" of a sensor node). The explorer dynamically adapts its content to the available devices and is implemented as an AJAX application designed to minimize the connections to the nodes while looking for a service.

C. Integration with Enterprise Applications

The real benefits of using a REST architecture are to be seen at the level of the enterprise-side integration, where the mashup is built. In our prototype, the integration was done with SAP MII, a commercial business solution used in the manufacturing industry. SAP MII enables non-programmers to visually combine data sources such as Web Services, databases, XML

documents with higher level business applications such as an ERP (Enterprise Resource Planning) application, by simply drag and drop building bricks from a palette. In this respect we can consider MII as a mashup editor, meant for end-users to create small applications useful for their particular business.

Let us imagine a composite application that reports the temperature of a shipment (to which a SunSPOT was attached) to an ERP application. We briefly expose the simplified steps required for building such an application comparing an implementation that uses Web Services and RESTful services, emphasizing on the conceptual differences.

The first step is to identify the temperature service on the SunSPOT. In the WS-* world this is usually done by searching for the previously registered service on a UDDI (Universal Description, Discovery and Integration) server. For real-world services this process is rather complicated since we are looking for one particular service on one particular device, meaning context (and in particular location) is of prime importance. For RESTful services the identification method is exploratory. Thanks to the links between the resources the user finds the service by clicking links (with the Resource Explorer, see Figure 2, or plain HTML pages in a web browser) as she would browse web pages, eventually finding the correct service.

Once the service is identified, its interface needs to be used. For the WS-* service the interface is the WSDL file. Once retrieved, this WSDL file can be fed to a Web-Service building brick which generates an invocation class to be used when consuming the service (a stub). During our experiments this process was error-prone because WSDL files, although standardized, often contain proprietary subtleties that make them hard to parse for MII. For RESTful services the URI (http://.../spot1/sensors/temperature) of the resource providing the service along with one of the HTTP verbs (e.g GET) is the interface. Thus, this URL can be fed to an HTTP building brick of MII without the need for it to generate anything like a stub. One of the direct advantages of this approach is that the service can be tested directly from the browser, by typing its URL, without the need for using a dedicated tool as for WS-* services. The next step consists of using an XML (in the case of WS-*) or a JSON (in the case of our RESTful implementation) processor to extract the actual data out of the service response. In the last step we feed this data to an ERP building brick taking care of mapping the correct ERP field to the temperature.

IV. Discussion and Future Work

In this paper we introduced several integration techniques for the use of embedded devices to create physical mashups within the enterprise. We especially focused on our experience with WS-* web services deployed on embedded sensors as well as the use of middleware solutions. While web services seem to be suitable for integration scenarios with a longer lifespan and strong QoS requirements it is also rather rigid and too heavy to be deployed on most embedded devices. Furthermore, due to the programatic complexity, WS-* or middleware approaches are not well-suited for empowering end-user to create ad-hoc applications. Thus, we proposed the use of a RESTful approach, implemented it on the SunSPOT plateform. Reusing the existing, successful and well-known standards of the web allows to make any physical object part of a "Web of Things", therefore directly addressable and usable using well-known tools.

The reported work also exposes a number of issues that require further investigation, as for example improving scalability of the architecture when concurrent requests occur on single nodes, better support for asynchronous communication of the sensed values, standards and models for the returned values. Finally, we are currently further evaluating the architecture, both in quantitative terms (e.g. overhead introduced by using HTTP and REST on the nodes) and qualitative terms (e.g. ease of integration and development).

Acknowledgment

The authors would like to thank the European Commission and the partners of the European IST FP6 project "Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices" (SOCRADES - www.socrades.eu) for their support.

References

[1] E. Fleisch and F. Mattern, *Das Internet der Dinge*. Springer, 2005.

[2] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *Proc. of IEEE Service Computing*, 2007, pp. 332–339.

[3] V. Hoyer, K. Stanoesvka-Slabeva, T. Janner, and C. Schroth, "Enterprise mashups: Design principles towards the long tail of user needs," in *Proc. of IEEE Services Computing*, vol. 2, 2008, pp. 601–602.

[4] M. Marin-Perianu, N. Meratnia, P. Havinga, L. de Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, and G. Stromberg, "Decentralized enterprise systems: a multiplatform wireless sensor network approach," *IEEE Wireless Communications*, 2007.

[5] L. M. S. de Souza, P. Spiess, D. Guinard, M. Koehler, S. Karnouskos, and D. Savio, "Socrades: A web service based shop floor integration infrastructure," in *Proc. of the Internet of Things Conference (IOT 2008)*. Springer, 2008.

[6] R. Dickerson, J. Lu, J. Lu, and K. Whitehouse, "Stream feeds - an abstraction for the world wide sensor web," in *Proc. of the Internet of Things Conference (IOT 2008)*, 2008.

[7] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, "Tinyrest - a protocol for integrating sensor networks into the internet," in *Proc. of the Workshop on Real-World Wireless Sensor Network (SICS)*, Stockholm, Sweden, 2005.

[8] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin, "pREST: a REST-based protocol for pervasive systems," in *Proc of the IEEE Conference on Mobile Ad-hoc and Sensor Systems*, 2004, pp. 340–348.

[9] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, May 2007.

[11] E. Zeeb, A. Bobek, H. Bohn, S. Prüter, A. Pohl, H. Krumm, I. Lück, F. Golatowski, and D. Timmermann, "WS4D: SOA-Toolkits making embedded systems ready for Web Services," in *Proceedings of the Open Source Software and Product Lines Workshop (OSSPL07)*, 2007.

[12] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *Proc. of the International Conference on World Wide Web (WWW 2008)*. ACM, 2008, p. 805814.