Mashing Up Your Web-Enabled Home

Dominique Guinard

[1] Institute for Pervasive Computing, ETH Zurich
[2] SAP Research CEC Zurich
dguinard@ethz.ch

Abstract. Many efforts are currently going towards networking smart things from the physical world (e.g. RFID, wireless sensor and actuator networks, embedded devices) on a larger scale. Rather than exposing real-world data and functionality through proprietary and tightly-coupled systems, several projects suggest to make them an integral part of the Web. As a result, smart things become easier to build upon and the scalability and evolvability of the Web can be leveraged. In this paper we look at "physical mashups", were tech-savvy create lightweight, ad-hoc applications on top of smart things just as they currently create Web 2.0 mashups. We present the early version of a mashup framework that provides services for composing things and illustrate this by means of two mashup editors.

1 Introduction

A seamless integration of smart things to the Internet brings them one step closer to the Web. Much is to gain from Web integration as it drastically eases the usually rather tedious development of applications on top of embedded computers.

The first step towards this goal is to bring things to the Internet, either directly using technologies such as IPv6 (lowpan) [1] or through translation gateways [2] (reverse proxies, in Web terms). Further, Web servers are needed on devices to truly achieve integration on the application, i.e. Web layer. Well established research in embedded Web servers and recent research in tiny Web servers makes this possible [3]. However, having a Web server running on an appliance is only the first step. Indeed, there is still a need for modeling access to the smart thing functionality in a Web-oriented manner [4]. Several projects, sometimes referred to as "Web of Things" projects, advocate using the REpresentational State Transfer (REST) architectural style for interfacing with smart things [2,5,6].

The Web enabling of smart things also delivers more flexibility and customization possibilities for end-users. As an example, tech-savvy, i.e. end-users at ease with new technologies, can also build small applications on top of their devices. Following the trend of Web 2.0 participatory services and in particular Web mashups, users can create applications mixing real-world devices such as home appliances with virtual services on the Web. This type of applications is sometimes referred to as *physical mashups* [2]. As an example, a HiFi system could be connected to Facebook or Twitter in order to post the songs one mostly listens to. On the Web this type of small, ad-hoc application is usually created through a mashup editor, e.g. Yahoo Pipes, which is a Web platform that enables people to visually create simple rules to compose Web sites and data sources.

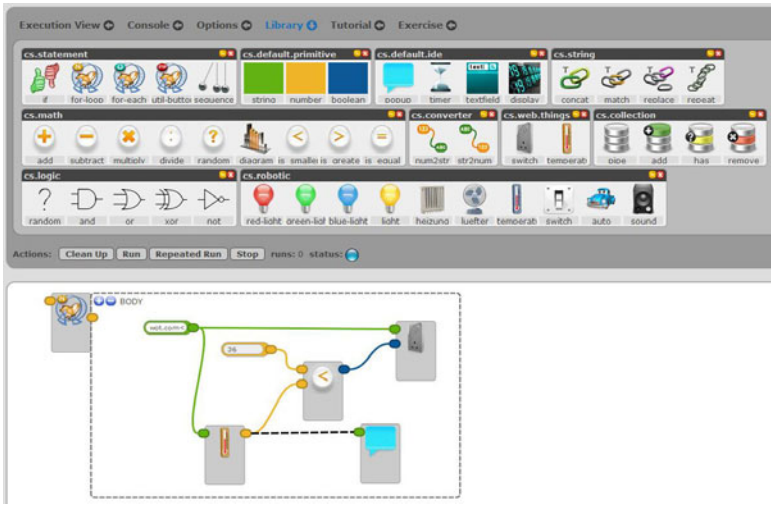

Fig. 1. A simple physical mashup using an adapted version of the Clickscript mashup editor

These concepts can now also be applied to empower users to create small applications on top of their smart things.

Our contribution in this paper is to outline a mashup framework that would support the creation of mashup editors for Web-enabled smart things.

2 Physical Mashups in the Web of Things

To illustrate this, we Web-enabled several devices. We first developed a Web API for Sun SPOT sensor nodes [7] which offers a broad range of sensors (temperature, acceleration, light) and actuators (leds, buttons, analog I/O ports). Requests for these sensors and actuators are formulated using URIs. For instance, typing a URL such as `http://.../spot1/sensors/temperature` in a browser, requests the resource *temperature* of the resource *sensors* of *spot1* with the HTTP method *GET*. Similarly, we Web-enabled the Ploggs energy meters[1] [2], which allows us to control (turn on/off) and monitor the energy consumption of plugged appliances (e.g. a lamp) in a RESTful manner. Further details about the Web-enabling process is provided in [2]. To better understand the concepts and requirements of a physical mashup editor, we began by testing and extending an existing mashup framework: the Clickscript project[2]. Clickscript is a Firefox plugin written on top of an AJAX library that allows people to visually create Web mashups by connecting building blocks of resources (Web sites) and operations (greater than, if/then, loops, etc.). Since it is written in Javascript, it cannot use resources based on low-level proprietary service protocols. However, it can easily access RESTful services. Thus, it is straightforward to create Clickscript building

[1] www.plogg.co.uk
[2] www.clickscript.ch

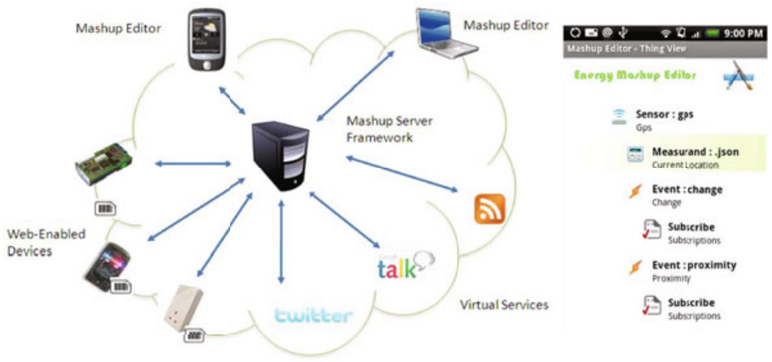

Fig. 2. Architecture of the physical mashup framework and screen-shot of a mobile mashup editor

blocks representing smart things. We used this approach to create building blocks for all the sensors and actuators of the Ploggs' and Sun SPOTs' RESTful APIs. As an example, we created a mashup shown in Figure 1. This mashup gets the room temperature by `GET`ting the temperature resource of a RESTful Sun SPOT. If it is smaller than 36 degrees Celsius, it turns off the air-conditioning system plugged to a RESTful Plogg.

2.1 Requirements of a Physical Mashup Platform

While this integration with Clickscript worked well, the tool is not meant for creating physical mashups and has some shortcomings that helped us to identify a number of requirements for a physical mashup platform:

R1: Support for several UIs and modalities: Since interactions in the physical world usually occur beyond the desktop metaphor, it is hard to imagine creating a "one-size-fits-all" mashup editor. We rather suggest creating a mashup framework on which multiple editors (e.g. mobile, tablet, etc.) can be built. This also lets users create their mashups locally, e.g. on a mobile phone, and export them to a more robust framework for execution.

R2: Support for describing things: Manually creating building blocks for each smart thing is not scalable to the diversity of the physical world. Thus the need for the mashup framework to support partially automated integration (aka service discovery) of things.

R3: Support for event based mashups: While the pull-based interaction model of the Web is fine for controlling smart things, it is suboptimal for monitoring them (e.g. sensors). Thus, the mashup framework should be able to handle events coming from smart things.

2.2 A Framework for Physical Mashups Editors

Based on these requirements we present our first prototype of a mashup framework for the Web of Things. As shown in figure 2, the system is composed of four main parts. First, we have Web-enabled devices and appliances such as the RESTful Ploggs and

Sun SPOTs. In our prototypes, we tag them with small 2-D bar-codes (QR codes) in order to ease the identification of their root URL with mobile phones. We then have “virtual” services on the Web such as Twitter, Google Visualization API, XMPP, etc. In the middle, the mashup server framework allows to compose services of different smart things as well as virtual services on the Web. It is in charge executing the work-flows created by end-users in their mashup editors. It discovers, listens, and interacts with the smart things over their RESTful API. The last component are the mashup editors. These applications allow users to create their mashups in a very easy manner. The mashup server framework is based on an open-source workflow engine called Ruote[3]. We extended Ruote in several ways in order to support our requirements. First, to fulfill R2, we created an RDFa schema and a compound Microformat that enables the discovery of smart things. Given the root URL of a smart things, the framework uses the RDFa or Microformat data to create a re-usable component representing the things’ functionality. Then, by using Web Hooks (i.e. callback URLs), we enabled smart things to push data to the mashups, fulfilling R3. Finally, we created a RESTful API that allows creating mashup editors on top of the framework, fulfilling R1.

2.3 Energy-Aware Mobile Mashup Editor

In order to prototype the use of the framework, we created a mashup editor on the Android Platform. The mobile editor offers wizards for creating simple mashups that are then exported to the mashup framework. Users first have to select the smart things they want to include in the mashup. They do this simply by scanning the things’ bar-codes using the phone’s camera. These codes are pointing back to the root URLs of the smart things’ RESTful APIs where the RDFa or Microformat description of the device and its services can be found. Users then setup the rules they want to enforce and the virtual services they want to interact with. As an example, we created a mashup that switches on appliances, e.g. turning the heating up, whenever your phone detects that your are coming home (based on your GPS trace). The right part of Figure 2 shows one of the wizard screens used to create this mashup.

2.4 Conclusion and Future Work

In this paper we introduced the idea of physical mashups and illustrated it by presenting the early prototype of a framework for creating editors of physical mashups. We briefly introduced a requirements for physical mashups. While we began addressing these, there is still work to be done to fully tackle them. As mentioned earlier, HTTP was designed as a client-server architecture where clients pull data. This interaction model works fine for control-oriented applications, however, monitoring-oriented applications are often event-based and thus Web-enabled smart things should also be able to push data to mashup frameworks, rather than being continuously polled. Overcoming the client-server architecture is now a core research topic in the Web community. Standards such as HTML 5 (and its “Web Sockets”) are also going towards asynchronous bi-directional communication.

[3] openwferu.rubyforge.org/

Another major challenge for mashups in a Web of Things is search and discovery of smart things. Consider billions of things connected to the Web. Searching for the things to be included in mashups becomes very complicated. Thus, smart things also need to have means to describe themselves in order for users to find them and for mashup frameworks to be able to automatically create wrapper for them. How to describe a thing on the Web so that both humans and applications (e.g. mashup frameworks) can understand what services it provides? Here again, the recent developments in Microformats, RDFa and HTML 5 Microdata seem to be going in a promising direction to support describing tomorrow's Web of Things.

References

1. Yazar, D., Dunkels, A.: Efficient application integration in IP-based sensor networks. In: Proc. ACM of the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys), Berkeley, CA, USA (November 2009)
2. Guinard, D., Trifa, V., Wilde, E.: Architecting a mashable open world wide web of things. Technical report, ETH Zurich (2010), `http://tinyurl.com/wotarchi`
3. Duquennoy, S., Grimaud, G., Vandewalle, J.J.: The Web of Things: interconnecting devices with high usability and performance. In: 6th International Conference on Embedded Software and Systems (ICESS 2009), Hangzhou, Zhejiang, China (May 2009)
4. Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M.: People, places, things: web presence for the real world. Mob. Netw. Appl. 7(5), 365–376 (2002)
5. Drytkiewicz, W., Radusch, I., Arbanowski, S., Popescu-Zeletin, R.: pREST: a REST-based protocol for pervasive systems. In: 2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems, pp. 340–348 (2004)
6. Luckenbach, T., Gober, P., Arbanowski, S., Kotsopoulos, A., Kim, K.: 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)
7. Guinard, D., Trifa, V., Pham, T., Liechti, O.: Towards physical mashups in the web of things. In: Proc. of the 6th International Conference on Networked Sensing Systems (INSS), Pittsburgh, USA (June 2009)