# UbiSOA Dashboard: Integrating the Physical and Digital Domains through Mashups

Edgardo Avilés-López and J. Antonio García-Macías

Computer Science Department
CICESE Research Center
Km. 107 Carretera Tijuana-Ensenada
Ensenada, Baja California, México
{avilesl,jagm}@cicese.mx

**Abstract.** The current Web 2.0 stage of the Internet provided the basis for web-based communities and services aimed at collaboration and information sharing. Furthermore, Internet is now an application platform in which Web applications can be integrated to provide augmented services that could bring the basis for ubiquitous computing scenarios. Recently, the concept of mashups has been used to refer to applications built upon the integration and combination of public Web API's and data sources. Ubiquitous computing mashups go further by combining the functionality of both software and hardware components in an attempt to exploit computation and services provided by everyday objects. Typically, developing a mashup requires highly specialized knowledge in many topics (such as using different programming interfaces and languages). This problem is greatly magnified in developing mashups of both physical and digital services due to the various integration and communication issues. We exemplify these concepts through the use of UbiSOA Editor, a system that allows the creation of ubiquitous computing mashups through simple activities such as dragging and dropping graphical representations of the involved services in a desired scenario. Then we talk about the planning and execution of a sample scenario as a showcase of what can be easily accomplished.

## 1 Introduction

The vision of ubiquitous computing, as articulated more than 15 years ago in a seminal article by Mark Weiser stated that "the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" [1]. Although this vision has not completely become a reality, we are getting closer. Nowadays we have GPS-equipped cars that will inform us about road conditions and propose driving routes, we can check our e-mail, make a phone call, listen to music and watch video clips with the same device (*e.g.*, Apple iPhone[1], which can be used for people-centric sensing applications [2]). All this is possible by micro-controllers embedded in the artifacts that we use everyday, some of which have truly embedded computers that we use without being aware of it. That is,

[1] Apple iPhone: http://www.apple.com/iphone/

technology is weaving itself into the fabric of everyday life. Weiser also stated that "machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking a walk in the woods". Although using computers, embedded or not, is currently not as refreshing as a walk in the woods, there have been some advances in multimodal user interfaces promising better interactions that fit the human environment.

Some have suggested that integration is one of the main barriers to realizing the vision of ubiquitous computing [3]. Indeed, some of the appropriate infrastructural elements exist, and there have been significant advances in human-computer interfaces, but there are many ubiquitous computing scenarios still a matter of science fiction and not part of our daily lives [4]. We believe that an important factor is that, for the creation of the applications in these scenarios, we are accessing the infrastructural elements and integrating them at a very low-level. This forces developers to focus on unnecessary infrastructure-level details, instead of focusing on application-level issues.

In this article we start by discussing two of the key issues in the development of ubiquitous computing applications: access and integration (Section 2). We then present UbiSOA and discuss how it can help in accessing context providers (Section 3). We also discuss the integration issue and talk about the Web as a platform for ubiquitous computing by showing how mashups can be helpful for integrating different services and primary context sources. We exemplify those concepts through the development of a ubiquitous computing mashup application by using of UbiSOA Editor (Sections 4 and 5). Finally, we present some conclusions and future work (Section 6).

## 2 Access and Integration

Our current technological situation can be easily understood with an example: the development of a dashboard application where we can monitor what is going on at home, at the office, or at our grandparent's house. We would be interested to check out if some lights were left turned on at home, or maybe turn them on while we are traveling to deter burglars from breaking-in. Maybe we would like to keep an eye on our grandparents and monitor the air quality at their home to warn us of possible gas leaks, or want to have an overview of their motion patterns during the day. It would also be useful if this dashboard could integrate some services we already use, like traffic alerts to avoid the bottlenecks in the highway. All the elements to make this a reality exist: there are already in the market Zigbee-enabled[2] devices to control lights in a networked fashion; many networked sensors, including environmental monitors and motion detectors, are also in the market; traffic alerts services are also available from many providers. But there are two key problems that make creating this dashboard difficult: access and integration.

First, let us examine access. The intelligent network-controlled lights may use the Zigbee protocol stack. This implies having to learn some Zigbee concepts, including how to bind end-devices to controllers, how to create a network, etc. Coding the application may force the programmer to use a particular development environment

[2] Zigbee Alliance: http://www.zigbee.org/

(*e.g.*, Codewarrior[3] for HCS08 microcontrollers) and programming language (typically C). For the sensor network at our grandparent's home, there are already many wireless sensor network kits available. TinyOS [5] is a very popular platform in these kits, but it imposes programming in a particular programming language called nesC [6] which, not being a general purpose language will be unlikely already known by a developer. This language has a certain philosophy where everything is a component and in order to build an application we have to bind them using certain interfaces. Other embedded systems concepts, such as threads of execution and atomic sections may be in order. The part developers would be more familiar with is the traffic alerts parts, since companies like Google and others have public APIs[4] that can be used to invoke their services by means of familiar Web concepts. Now that we have access to the different platforms (the home automation network, the monitoring network, and the traffic alerts), there is yet another problem: the integration of them all. Since the dashboard must be accessed anytime, the home automation network and our grandparent's sensor network must be properly Internet-enabled. This means, writing some sort of Internet gateway for each of them, and since their code is written in different languages and they have different idiosyncrasies, reutilization is mostly not possible. A web page could be the integration element, since the gateways can be configured to output HTML and some JavaScript code would allow to access the services provided by Google Maps[5].

## 3 Better Access with UbiSOA

The concept of the Ubiquitous Web (UW) [7] has received a lot of attention lately, as it defines a pervasive web infrastructure where physical objects are integrated into the world of web information and services. Many features make the Web attractive as a platform for ubiquitous computing: it is device-independent and language-agnostic, it offers low barriers to entry, it has a content-centric model and it is, for all practical purposes, ubiquitous.

Regarding the access problem, providing better abstractions to developers has been a long-time motivation in software engineering. Therefore, we have witnessed advances in programming methodologies and paradigms ranging from sequential programming, modular programming, object oriented programming, component-based programming, and more recently service-oriented programming. This recent approach arises in response to modern needs and complexities such as distributed software, application integration, as well as heterogeneity in platforms, protocols and devices, including Internet integration. It features a very high abstraction level; any functionality is exposed as a service, so the user just asks for a given service without knowing or having to deal with any low-level issues required for the functionality to be completed. For example, if we have just installed a new wireless sensor node, and we want to know what the moisture reading is, we would have to go through the cumbersome tasks discussed earlier. In contrast, with the service-oriented model, we would

[3] Codewarrior Development Tools: http://www.freescale.com/codewarrior/
[4] Google APIs: http://code.google.com/apis/
[5] Google Maps: http://maps.google.com/

just ask what services the network provides and then we would invoke the service for reading the moisture level.

This is precisely the philosophy behind UbiSOA (which is a work in progress and has evolved from TinySOA [8]). As other authors [9], we think that the service-oriented and context-driven models are excellent means to implement successful ubiquitous applications. UbiSOA is a platform comprised of many different services, resources and tools to allow the creation of ubiquitous applications. Instances of such services are localization, environmental primary context providers (wireless sensor networks), identification context (RFID devices), and others. The access to those services is driven through standard and easy to use Web services (via the SOAP and REST protocol) and Web feeds (Atom and RSS feeds). An important advantage of Web services and feeds is that developers can use the same tools and languages they normally use for developing applications. This means, no hardware specifications needed, no need to learn or be tied to a particular language, and no wasting valuable time with these issues.

## 4 Better Integration with Mashups

In its origins, the Web provided a way to easily share information. Shared contents consisted of simple hyperlinked documents hosted on Web servers and were only maintained by their authors. One can refer to this stage as the Web 1.0, where the Internet's available content was on a “read-only” mode. As technologies evolved, server-side scripting languages (such as Perl, PHP, and others) gained attention. This made the Web servers to start changing dynamically the information they provided. This new “read-write” paradigm, gave the basis to web-based communities and services aimed at collaboration and information sharing. In this paradigm the user is no longer a consumer; he continuously contributes with new information (as seen on wiki systems). This still current stage is known as Web 2.0, where Web sites typically include technologies such as AJAX-enabled communications, syndication through RSS feeds, and a clear adoption of standards on presentation and content handling. Furthermore, we can see Internet as a platform in which many different Web applications are now providing public available APIs to manage and use the information they provide. This led to the introduction of mashups.

The mashup concept originated in music, where fragments of two or more songs are mixed and rearranged together creating new songs or compositions. A mashup in computer science refers to applications created by the integration and combination of services and content from different Web public APIs and sources. A very common example is a mashup that uses Google Maps to show the location of the most recent pictures of a Flickr[6] account. There are even many tools online that can help you on creating mashups: Microsoft Popfly[7], Yahoo! Pipes[8], Google Mashup Editor[9] and others. Those systems are ready to integrate information from some of the most used

[6] Flickr: http://www.flickr.com/
[7] Microsoft Popfly: http://www.popfly.ms/
[8] Yahoo! Pipes: http://pipes.yahoo.com/
[9] Google Mashup Editor: http://editor.googlemashups.com/

Web applications that already provide access to their functionality through Web services or other standardized means.

Recently, the term ubiquitous computing mashup is being used referring to the attempt to move computation off the desktop and integrate it with the artifacts of everyday life [10]. Ubiquitous computing application development can highly benefit from mashup technology. Common functionality can be isolated into single and independent components (service providers) that can be later combined to accomplish functionality needed for a specific scenario. Also, as the number of devices and other data sources grows, the need to filter the available information will be higher; mashups can then be used to delegate processing of the multiple sources (data or even functionality) generating a more manageable and scalable feed. Although, there is one problem, real-time needs could hardly be supported if they aren't carefully considered in the design and operation of each one of the service providers. Other aspects of Web 2.0 technology are being used to power ubiquitous scenarios, one of them is the model of sharing and annotating physical augmented devices in the same way wiki systems are [11].

Since UbiSOA's services allow us to retrieve primary context information from sensor networks and RFID devices (environmental and identification), we can go back to the monitoring dashboard we discussed earlier; we will see how UbiSOA can make building this type of applications a lot easier.

In order to showcase what can be easily accomplished, let us introduce the following scenario: let's suppose there are multiple sensor-networks deployed in some locations and that there are RFID readers in the door of multiple rooms on different locations that users use to report their arrival and departure. We need a Web application in which we can easily browse through all these locations (ideally using a map) and see the current status for each one, also, we need to be aware of many events including a user arriving/leaving a location. Clearly, developing this without using third-party services would be highly complicated, mainly because of dealing with the communication and data extraction of the many different involved devices, besides the problem of how to combine and remotely access all the data.

Second, we need to know what services we can use to accomplish the scenario and what we are going to integrate. In our case we will use Google Maps handled by an AJAX interface (for driving and selection of multiple locations), sensor networks (to fetch sensor data) and RFID services (to track people arriving/leaving locations and possibly other objects). This is going to be a static mashup; all the different providers will be manually selected to build the final application. Mashups can also be dynamic, allowing choosing on execution time which services could be used. This is highly useful, because for instance, we could use a backup service in case the main one is down or too busy to obtain timely responses.

Before discussing the third step which is the implementation of the scenario, we need to introduce UbiSOA Editor. In our infrastructure, a mashup application can be coded by means of a script that specifies the services involved, the connections between them, and the data flow though the configuration or assembled by using the UbiSOA Editor, a system to create mashups through simple activities such as dragging and dropping graphical representations of the involved services to accomplish the specification of configurations that support a desired scenario. In our scenario of use, a user must carry a device that provides him with personal services (such as localization, contacts, notification, and others), runs mashup applications, and allows

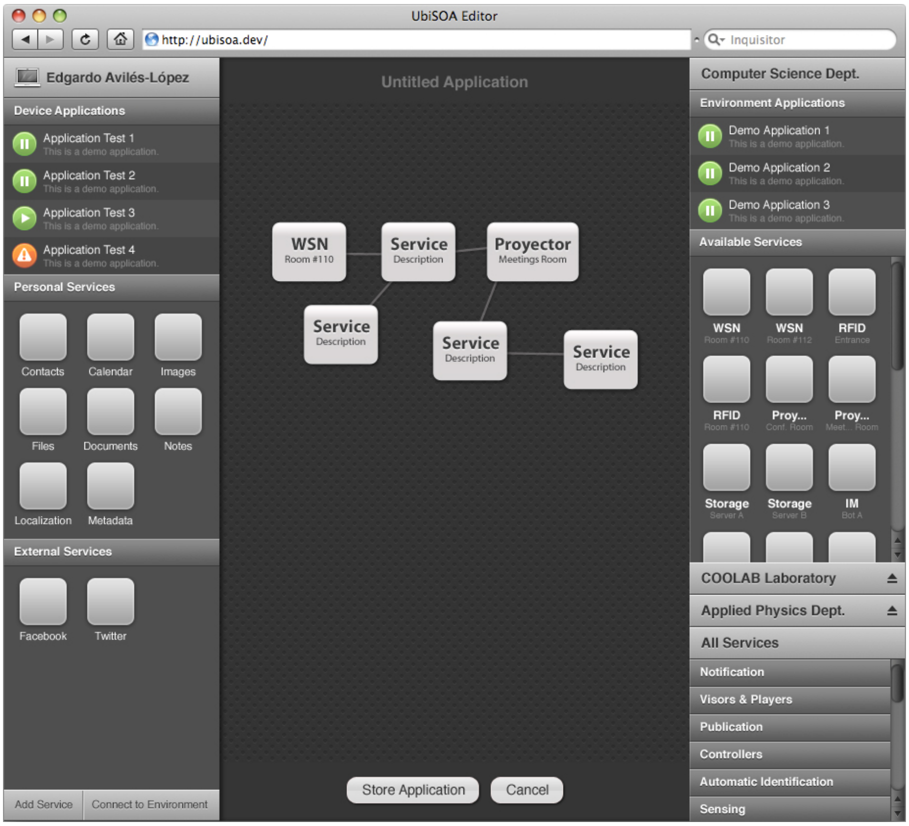


**Fig. 1.** Overview of the UbiSOA Editor

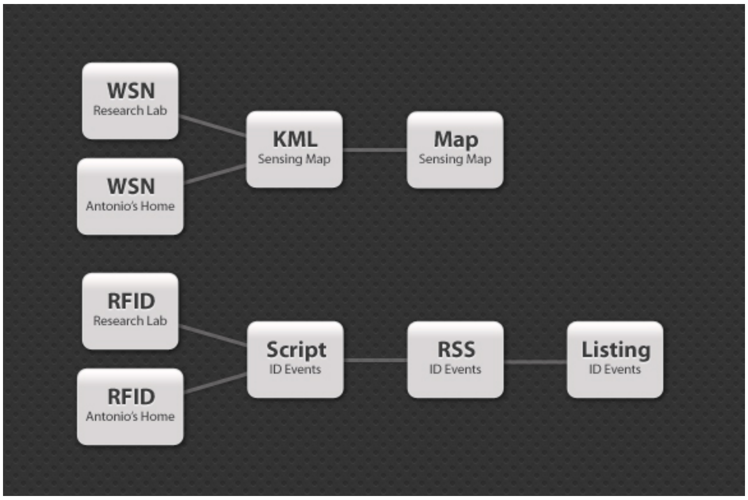
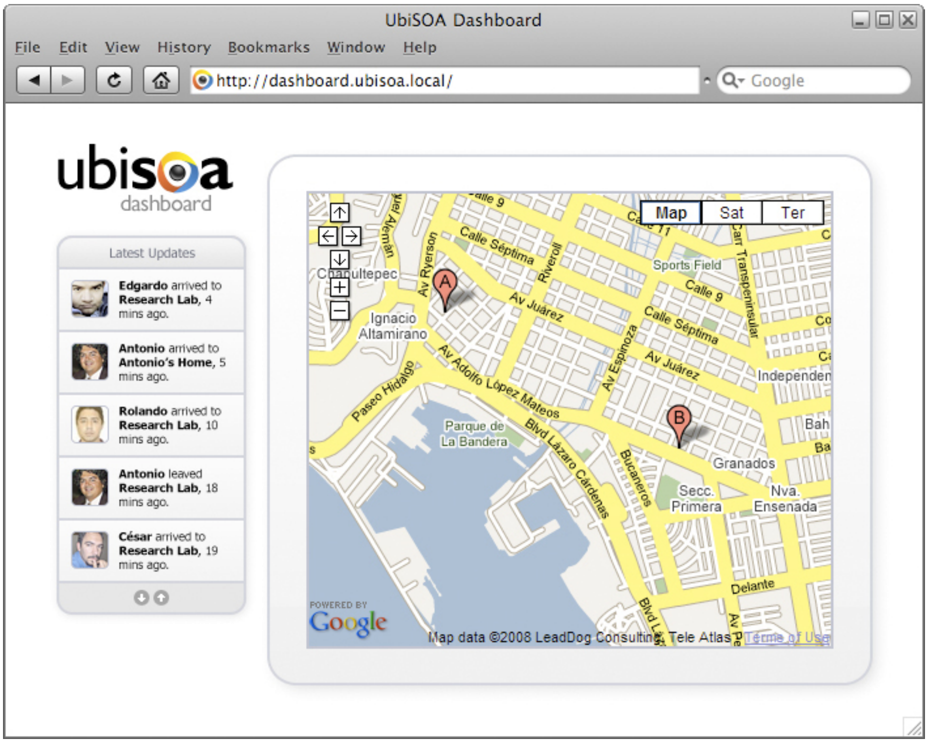


**Fig. 2.** Mashup configuration for the dashboard application

access to the infrastructure through the UbiSOA Editor. The personal applications and services are shown on the left sidebar of the editor (see Fig. 1). The UbiSOA infrastructure automatically discovers the services that are provided on the user environment and the applications that are currently running (those are shown on the right sidebar). Additionally to the basic personal services, the user can add external services such as his Facebook or Twitter account. The editor also allows interaction with remote environments. The area at the center of the editor is the instantiation area where the services are dropped, interconnected, and then stored into an environment or in the personal device as mashup applications.



Edgardo arrived to Research Lab, 4 mins ago.

Antonio arrived to Antonio's Home, 5 mins ago.

Rolando arrived to Research Lab, 10 mins ago.

Antonio leaved Research Lab, 18 mins ago.

César arrived to Research Lab, 19 mins ago.

**Fig. 3.** The dashboard application build upon the embedded code from the mashup

Continuing with the dashboard implementation, we just have to drag, drop and interconnect the appropriate services for the scenario we desire. We need a sensor network service and a RFID reader service for each of the environments we want to monitor (we will first need to connect to them) and utility services that will provide our visualization needs. Is important to note that UbiSOA doesn't just introduce custom services, it also allows the integration with current Web resources and APIs such as Google's. In our sample dashboard we are going to support 2 locations, each with a WSN and a RFID service. As utility services, we need a KML maker service and a Script service that runs custom code for interpreting the RFID readings received from the readers, once interpreted we send the results to a RSS maker service. We could already use the generated KML and RSS files as data feeds for our dashboard application but UbiSOA also provides tools to embed special HTML+JavaScript code in our web applications that will help us on visualizing and updating those files when an infrastructure event is generated. Those tools are also services, in this case the Map and Listing services. The full configuration of this mashup is shown in Fig. 2. Once interconnected, we need to store this application in the environment or in our device to finally get the embedded codes of the sensing map and the identification listing. Figure 3 shows the finished dashboard application.

## 5 Towards the Ubiquitous Web

Almost by accident, or at least without being part of a grand plan, the Internet is playing a very important role in making the vision of ubiquitous computing a reality. What started as a medium for publishing and sharing information is now seen as a good platform for interconnecting all types of physical devices, integrating them into the information universe. The next stage in the evolution of the Web, already in progress and called by some Web 3.0, will be realized with semantic web developments. But beyond that, in what is called the Ubiquitous Web or the Internet of Things ([12, 13]), we need better means for accessing the services provided by physical objects and for integrating these services. Current Web 2.0 technologies, in the form of Web services for access and mashups for integration, provide a step ahead, although can only be seen as transitional technologies.

A decade ago, Weiser considered the Internet as a transitional stage between PC and UbiComp scenarios [14]. We can see proofs of this transition in current works such as the seamless integration of wireless sensor networks with IP networks [15]. More importantly, not only we have witnessed the needed changes on infrastructure and services, there are also changes on social aspects too; users are more prone to use a Web-based tool to do their tasks, for instance, increasingly more people are using webmail providers than institutionally hosted mail services, another notorious example is the adoption of instant-messaging systems.

## 6 Conclusions and Future Work

We believe that the benefits of mashup applications are important; they allow us to fulfill tasks and rapidly construct proof of concept applications which is highly valuable for ubiquitous computing research. Although there is a problem to overcome: developing a mashup application requires Web programming expertise due to the management of data representations, connectivity, and standards [16]. Higher-level abstraction tools that assist in constructing ubiquitous mashups are needed. These tools need to hide from the developer issues such as security, event management, real-time needs, communication, and others.

UbiSOA is still a work in progress; we are currently exploring communication and event notification models to allow a more transparent integration and cooperation between service providers. In this paper we have introduced the UbiSOA Editor which is an important part of the UbiSOA infrastructure. We plan to further improve the design by conducting a usefulness and adoption of technology experiment.

**Acknowledgements.** Financial support for this project was provided by the Mexican Council for Science and Technology (CONACyT).

## References

1. Weiser, M.: The computer for the 21st century. In: SIGMOBILE Mob. Comput. Commun. Rev., vol. 3, pp. 3–11. ACM Press, New York (1999)
2. Miluzzo, E., Oakley, J.M.H., Lu, H., Lane, N.D., Peterson, R.A., Campbell, A.T.: Evaluating the iPhone as a mobile platform for people-centric sensing applications. In: Proc. of Intl Workshop on Urban, Community, and Social Applications of Networked Sensing Systems (UrbanSense 2008) (November 2008) (to appear)
3. Davies, N., Gellersen, H.: Beyond prototypes: Challenges in deploying ubiquitous systems. In: IEEE Pervasive Computing, Piscataway, USA, vol. 1, pp. 26–35. IEEE Educational Activities Department (2002)
4. Schmitz, M., Endres, C., Butz, A.: A survey of human-computer interaction design in science fiction movies. In: INTETAIN 2008: Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment (January 2008)
5. Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D.: TinyOS, an operating system for wireless sensor networks. Springer, New York (2005)
6. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC language: A holistic approach to networked embedded systems. In: PLDI 2003: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pp. 1–11. ACM, New York (2003)
7. W3C: Workshop on the ubiquitous Web (2006)
8. Avilés-López, E., García-Macías, J.: Providing service-oriented abstractions for the wireless sensor grid. In: Cérin, C., Li, K.-C. (eds.) GPC 2007. LNCS, vol. 4459, pp. 710–715. Springer, Heidelberg (2007)
9. Yang, H., Jansen, E., Helal, S.: A comparison of two programming models for pervasive computing. In: SAINT-W 2006: Proceedings of the International Symposium on Applications on Internet Workshops, Washington, DC, USA, pp. 134–137. IEEE Computer Society, Los Alamitos (2006)
10. Hartmann, B., Doorley, S., Klemmer, S.R.: Hacking, mashing, gluing: Understanding opportunistic design. Pervasive Computing 7(3), 46–54 (2008)
11. de Ipiña, D.L., Vázquez, J.I., Abaitua, J.: A web 2.0 platform to enable context-aware mobile mash-ups. In: Schiele, B., Dey, A.K., Gellersen, H., de Ruyter, B., Tscheligi, M., Wichert, R., Aarts, E., Buchmann, A. (eds.) AmI 2007. LNCS, vol. 4794, pp. 266–286. Springer, Heidelberg (2007)
12. Broll, G., Siorpaes, S., Rukzio, E., Paolucci, M., Hamard, J., Wagner, M., Schmidt, A.: Supporting mobile service usage through physical mobile interaction. In: PERCOM 2007: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications, Washington, DC, USA, pp. 262–271. IEEE Computer Society, Los Alamitos (2007)
13. Michahelles, F., Thiesse, F., Schmidt, A., Williams, J.R.: Pervasive rfid and near field communication technology. IEEE Perv. Computing 6(3), 94–96 (2007)
14. Weiser, M., Brown, J.S.: The coming age of calm technology. Beyond Calculation: The Next Fifty Years of Computing 75(85) (1997)
15. Hui, J.W., Culler, D.E.: IP is dead, long live IP for wireless sensor networks. In: Proceedings of the 6th international Conference on Embedded Networked Sensor Systems (SenSys 2008) (November 2008)
16. Wong, J., Hong, J.: Marmite: end-user programming for the web. In: CHI 2006: CHI 2006 extended abstracts on Human factors in computing systems, pp. 1541–1546. ACM, New York (2006)