Sharing Using Social Networks in a Composable Web of Things

Dominique Guinard, Mathias Fischer, Vlad Trifa
Institute for Pervasive Computing, ETH Zurich
and SAP Research CEC Zurich
8092 Zurich, Switzerland
Contact Email: dguinard@ethz.ch

***Abstract*—In the emerging "Web of Things", digitally augmented everyday objects are seamlessly integrated to the Web by reusing Web patterns such as REST. This results in an ecosystem of real-world devices that can be reused and recombined to create new ad-hoc applications. This, however, implies that devices are available to the world. In this paper, we propose a platform that enables people to share their Web-enabled devices so that others can use them. We illustrate how to rely on existing social networks and their open APIs (e.g. OpenSocial) to enable owners to leverage the social structures in place for sharing smart things with others. We finally discuss some of the challenges we identified towards a composable Web of Things.**

I. Introduction and Related Work

Pervasive computing has been dealing to a large extent with the integration of digital artifacts with the physical world. The "Internet of Things" has explored the development of applications built upon various networked physical objects [1]. Items in the physical world such as sensor and actuator networks, embedded devices, appliances and everyday digitally enhanced objects (subsequently called *smart things*) are still mainly grouped into small islands disconnected from each other. Increasingly, consumer electronics possess Internet connectivity (as for example the Chumby or Nabaztag), however cannot be controlled and monitored without dedicated software and/or proprietary protocols. As a consequence, smart things are hard to integrate with each other and this prevents the realization of an ecosystem on top of which composite applications can be easily developed.

A new breed of applications and research projects propose the integration of physical things with the Web [2], [3], [4]. Usually unified under the umberella of "Web of Things" projects, they promote the integration of smart things not only to the Internet (i.e. at the network level), but also to the Web (i.e. at the application level). This enables Web tools and techniques (e.g. browsers, search engines, caching systems,...), languages (e.g, HTML, JavaScript, mashups) and interaction models (e.g. browsing, linking, bookmarking) to be extended to the real-world. In practice, these projects are built upon the REST architectural style [5] by embedding Web servers on smart things.

At large, the Web of Things could materialize into an open ecosystem of digitally augmented objects on top of which applications can be created using standard Web languages and tools. Just as tech-savvys create Web mashups by combining several Web resources [6], in a Web of Things they can create "physical mashups" by composing virtual and physical services [3], [2], [7]. The success of Web mashups is closely dependent upon the trend for Web 2.0 services (e.g. Google, Twitter, Wordpress, Doodle, etc.) to provide access to some of their services through relatively (often REST-based) simple open APIs (Application Programming Interface) on the Web. Mashup creators in turn often share their mashups on the Web (sometimes through directories such as Mashable[1]) and expose them through open APIs as well, making the ecosystem grow with each application and mashup.

Enabling this model on a Web of Things requires a sharing mechanism for smart things, by enabling access to the services offered by devices through their API. For example, one could share the energy consumption sensors in his house with the community. However, this is a complex process since these devices are part of our everyday life and their public sharing might result in serious privacy violations. HTTP already provides authentication mechanisms (e.g. HTTP Authentication[2]) based on credentials and server-side groups. While this solution is already available for free on most (embedded) Web servers it presents a number of drawbacks. First, when considering a large number of smart things it becomes quite unmanageable to share credentials for each of them. Then, as the shared resources are not advertised anywhere, sharing also requires the use of secondary channels such as sending emails containing credentials to people. Several platforms such as SenseWeb [8] or Pachube[3] propose to overcome these limitations by providing a central platform for people to share their sensor data. However, these approaches are based on a centralized data repository and do not allow direct interaction with smart things.

A promissing solution would be to leverage existing social structures to allow sharing of things, and for that we build upon social networks (e.g. Facebook, Linkedin, Twitter, etc.) and their (open) APIs. Using social networks enables users to share things with people they know and trust (e.g. relatives, friends, colleagues, fellow researchers, etc.), without the need

[1] http://mashable.com
[2] http://www.ietf.org/rfc/rfc2617.txt
[3] http://www.pachube.com

to recreate yet another social network or user database from scratch on a new online service. Additionally, this enables advertising and sharing through a unique channel: you can tell your friends about the sensors you shared with them by automatically posting messages to their profile or newsfeed. The SenseShare project [9] goes towards this direction as it allows users to share sensor data with their friends. It also allows owners to apply different filters to the data before sharing it. However, similarly to Pachube, SenseShare acts as a datastore between the sensors and the clients. It allows sharing the data coming from sensors but does not support direct interactions with the sensors. Furthermore, SenseShare uses Facebook only as social network. Such a tight coupling with a single external service whose contract (API and allowed accesses) is subject to change over time, is problematic. Indeed, We want to support different social networks, and enable users to control which one to use for each device.

In this article we propose a system to share things and facilitate access to real-world services offering a RESTful Web API. Our core contribution is a Web platform called Social Access Controller which:

- Acts as an authentication and sharing proxy for smart things helping users to fine-tune the nature of interactions they want to allow for their smart things (e.g. read-only, read-write, etc.).
- Manages access control based on the existing social structure of several social networks in order to enable smart things owners to share with people they know and trust.
- Uses social networks for advertising shared smart things.

The Social Access Controller is described in Section III. Its implementation is briefly described in Section IV. Finally, in Section V we discuss the challenges identified while designing and implementing this work.

The Social Access Controller requires for smart things to offer a Web API as described in Web of Things related projects [10], [3], [2], [4]. Thus, we begin by briefly introducing our view of a Web of Things architecture.

II. Integrating Smart Things to the Web

To share smart things via the Web, we first need seamlessly integrate them to the Web. This is achieved in two steps. First, smart things are connected to the Internet by embedding Web servers directly onboard, as suggested in [11], [12], [10]. Then, the functionalities of smart things are exposed as resources by applying the REST architectural style [5]. REST is core to the Web and focuses on creating loosely coupled services so that they can be easily reused. It uses URIs for encapsulating and identifying services on the Web and HTTP as a uniform application interface. It also decouples resources from their presentation and provides mechanisms for clients to select the best possible formats. This makes REST an ideal candidate to build an "universal" API for smart things [2].

Despite the recent developments in embedded Web-servers [11], [13], we cannot assume that all smart things will be directly connected to the Internet and offer RESTful interfaces. In a number of cases (e.g. RFID tagged objects), it makes sense to expose services of devices with limited capabilities as (RESTful) resources through an intermediary component that understands the device specific protocols. Thanks to the concept of intermediaries (e.g. proxies) in RESTful architectures, such a design can be easily implemented. Such a proxy – called **Smart Gateway** – that bridges the Internet with a certain number of device specific protocols (e.g. Bluetooth, Zigbee) has been proposed in literature [14] and we use it as basic building block for our system.

A. Resource Oriented Energy Meters

We briefly describe the concrete implementation of these concepts with an energy meterting sensor nodes further described in [2]. For this implementation we used a smart power plug called Plogg[4] that measures the electricity consumption of the devices plugged into them, and communicates it over Bluetooth or Zigbee. However, the integration interface offered by the Ploggs is proprietary, which makes the development of the applications using Ploggs rather tedious.

We implemented a Smart Gateway for the Ploggs that finds automatically all the Ploggs by scanning the environment for Bluetooth devices. Then it makes the discovered Ploggs available as RESTful resources. As an example, an HTTP call on http://.../EnergieVisible/SmartMeters/ with the `GET` method. returns the list of all the smart meters (i.e. Ploggs) connected to the gateway in the JSON[5] format. Similarly, requesting the `PUT` method on http://.../LampMeter/status alongside with the HTTP payload `status=off`, can be used to turn the lamp connected to the smart meter off.

III. An System for Sharing Smart Things

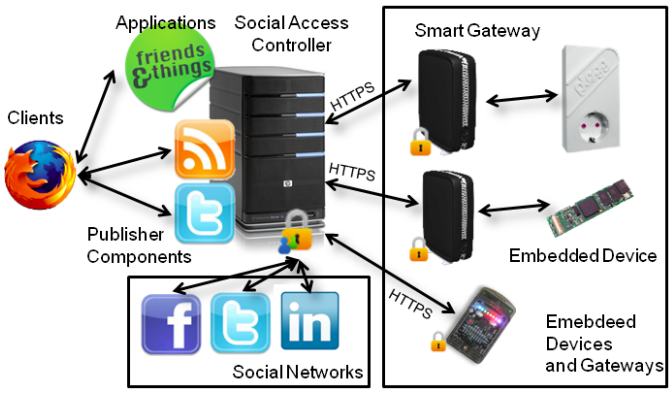

Fig. 1. Simplified component architecture of the Social Access Controller. SAC serves as authentication proxy between clients and embedded devices.

The system we proposed enables owners of smart things to share them on the Web easily. Consider for example the Resource Oriented Energy Meters described above. Sharing the energy consumption recorded by all these smart meters on the Web, enables the creation of very interesting applications.

[4] `http://www.plogginternational.com`
[5] `http://www.json.org`

For instance a mashup on a map could show the the consumption of each individual in a group of friends. To some extends existing platforms like Pachube allow the creation of such applications. However, sensor readings are only shared amongst user registered on these platforms. On the other hand, one might also want to share actuators. As an example, one can enable switching on/off devices by close relatives. Similarly, a Web-enabled Hi-Fi system can enable songs to be played remotely through a RESTful interface. Sharing it with close friends would enable them to remotely play songs for you.

The global architecture presented on Figure 1 addresses these use cases. It is composed of a central Web application called Social Access Controller which is in charge of creating the link between social networks and smart things. We describe the system starting with the requirements on smart things for them to be shareable. We then further describe how things can be shared by owners and used by social network members. We finally talk about how the system can be used to aggregate data coming from several smart things.

Note that we define owners as people owing or administrating smart things (e.g. the Resource Oriented Energy Meters) and trusted connections as the people owners share their smart things with (e.g. friends, colleagues or relatives). It is worth noting, however, that owners and trusted connections could also be applications.

A. Web of Things Constraints

The prerequisites for our sharing system are based on the Web of Things architecture briefly described before as well as the constraints of RESTful architectures [15]. We summarize them as follow:

- **Addressability:** All the shareable functions offered by smart things should be modeled as resources [5] which are addressable and identified by URLs.
- **Uniform Interface:** The actions available on resources should be compliant with the HTTP verbs (e.g. a GET on a resource retrieves a representation of that resource).
- **Resource Description:** The embedded Web servers on smart things (or Smart Gateways) should support the HTTP `OPTION` method which allows for retrieving the authorized methods for any URL. All the resources should be listed on HTML linked together so that they are crawlable [15].

Additionally, the direct access to smart things should be restricted as shown on the rightmost part of Figure 1. If not provided "out-of-the-box", this can be done by setting up the Web server at the device level or at the gateway level to accept only authenticated HTTPS traffic thus requiring credentials for any incoming request.

B. Leveraging Social Network APIs

SAC is an authentication proxy between clients (e.g. Web browsers) and the smart things. Rather than maintaining its own database or list of trusted connections and credentials – as it would be done with simple HTTP Authentication – it connects to a number of social networks to extract all potential users and groups one could share with.

This is possible as most social networks offer a Web API (e.g. Facebook Connect[6]). Providing an open Web API is one of the success factors of social networks themselves [16]. Indeed, these APIs allow third-parties Web applications to be built using partial data extracted from the social networks and thus to enhance the functionality of the social networks.

C. Leveraging Web Authentication APIs

Not only SAC needs to be able to retrieve lists of friends and relatives, it also needs a mechanism to authenticate owners and trusted connections. Owners need to be authenticated to their social networks so that they can access SAC and retrieve lists of trusted connections. Similarly, trusted connections need to be authenticated by SAC to confirm that they can access the shared smart things.

This is what open Web authentication APIs provide: delegated authentication on the Web. Using such APIs external applications such as SAC can request service providers such as social networks to authenticate one of their users. They can further query the service providers for profile information. Several standards compete for delegated authentication but OAuth[7] is certainly the most prevailing one and is implemented by several social networks and Web 2.0 services.

A delegated authentication for SAC presents two advantages. First, owners and trusted connections do not need special credentials for accessing SAC, as they can use the credentials of any social network or service on the Web that supports delegated authentication. Second, SAC does not need to hold profile information about the users (a user ID is enough) and can support several social networks for a single trusted connection or owner.

D. Sharing Based on Social Networks

The sharing process occurs in three phases:

1) First the owner accesses SAC by logging in using at least one of his social networks credentials. SAC then uses delegated authentication with the social network to identify the owner.
2) Then, the smart thing to be shared has to be crawled in order to identify what are the resources and verbs of its RESTful API, i.e. functionalities can be shared for that thing.
3) Finally, the user generates the access control list of the smart thing by selecting which trusted connections can interact with what resource.

a) Discovery by Crawling Things: Initially, the owner must inform SAC about the smart things he owns. He does this by providing the (HTTP Authentication) credentials to access a smart thing or the credentials of a Smart Gateway that manages several smart things. In our current implementation, smart things provide an HTML page describing and, most

[6] `http://developers.facebook.com/connect.php`
[7] `http://oauth.net/`

importantly, linking to their sub-resources. SAC uses the given credentials and URI to access the homepage and crawls it for offered resources (i.e. services). This allows SAC to "discover" the resources and sub-resources the owner can share for a given smart thing or Smart Gateway. Furthermore, the crawler extracts the operations one can execute on resources by calling the HTTP method `OPTION` for each resource. This method returns the methods supported for a particular URI, e.g. `PUT`, `POST`, `GET`, etc.

To illustrate this process, consider an owner who wants to share the RESTful Ploggs introduced before. The user gives the credentials to the Ploggs Smart Gateway alongside with its base URI, e.g. http://.../EnergyMonitor. The crawling engine will browse that page and detect the links to the sub-resources of the Ploggs such as:

```
/EnergyMonitor/load
/EnergyMonitor/ploggs/Kettle
/EnergyMonitor/ploggs/Light/status
```

For each resource, the crawler will also retrieves the HTTP methods it supports. For example, the `Kettle` resource only supports `GET` whereas the `status` resource also supports `PUT` to switch on or off the device. The result of this process is a list of resources that can be shared. An example representation of such as list is shown on Figure 2.

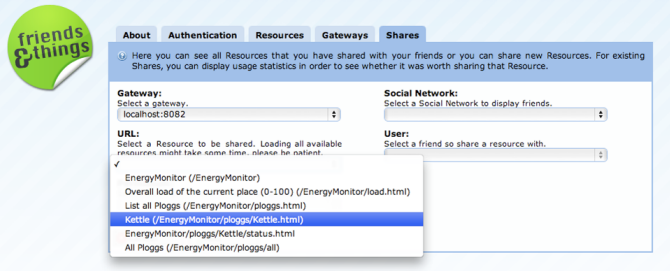

Fig. 2. Screenshot of the FAT (Friends and Things) application on top of SAC. The owner can select the smart things (i.e. resources) he wants to share with some of his trusted connections.

b) Sharing Things: The user can then share his smart things with friends, relatives, or colleagues (subsequently called trusted connections). SAC manages a list of all the social networks a user is currently logged in with. SAC then uses the open Web API of each social network and queries them for lists of friends and other connections as shown on the lower part of Figure 1. All these connections are then compiled into a global list of potential connections. The owner selects the connections he wants to share his smart things with. He can either share complete smart things (e.g. a Plogg) or their sub-resources only. Furthermore, he can choose to share resources in read-only (i.e. only with the `GET` verb) or read-write (i.e. giving access to all the available HTTP methods). Figure 2 shows the user interface to share things.

E. Accessing Shared Things

When an owner shares resources with a trusted connection, the latter is informed about it directly on his social network. In

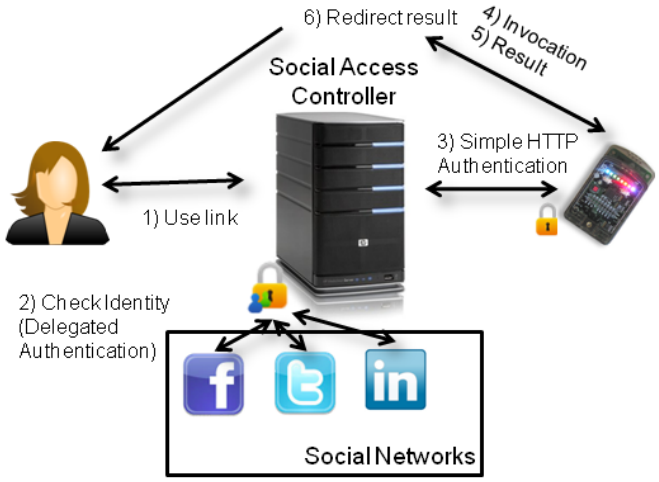

Fig. 3. Accessing shared objects using delegated authentication through the Social Access Controller.

the case of Facebook, it publishes a message to the newsfeed of the trusted connection as shown on the left part of Figure 1. In the case of Twitter it simply tweets a message to the trusted connection (e.g. "Rachel shared her Ploggs Energy sensors with you"). The posted messages also contains a *link* that redirects to the shared resource. To access it, the trusted connection can also login to SAC with his social network credentials to retrieve a list of smart things he is allowed to access.

The link does not point to the smart thing directly but to an instance of SAC which acts as the authentication proxy as shown on Figure 3. Whenever a potentially trusted connection uses the provided link (step 1 on Figure 3) SAC will check his identity using the delegated authentication (step 2) method we introduced before. If the potentially trusted connection logged in successfully with one of his social networks SAC will internally check whether this person also has access to the requested resource. If it is the case SAC logs on the shared resource using the credentials provided by the owner (step 3) when registering the resource. It then redirects the HTTP request of the trusted connection to the shared resource (step 4). Finally, it redirects the result (step 5) directly to the HTTP client of the trusted connection (step 6), e.g. to a Web browser.

F. Syndication of Shared Data

While direct access to a single device might be interesting for control scenarios, as for instance playing a song on a HiFi system or turning off a device remotely, monitoring use cases require a system that allows to group several events coming from smart things together and publish them to a messaging or feeds server on the Web.

SAC provides a syndication mechanism that can be used to monitor several smart things at once. It consists of a Publisher Component which periodically polls the smart things for updates and sends the updates to a syndication server (e.g. an ATOM server). Publishers can be parameterized by specifying the amount (number of characters or percentage of

change) of content that should be changed in order to generate a new event. A regular expression which should be satisfied can also be specified. Finally, another regular expression can be used to reformat the content of the event before publishing it.

An example scenario for this system could be a friend who can be informed when you leave work. By monitoring the energy consumption of your computer, a notification will be generated and transmitted when your computer is turned off. Another scenario could be a friend who creates a simple physical mashup with Google Maps that shows friends available for a drink in the neighborhood. This mashup could be simply based on an ATOM feed to which the Publisher Component sends update events whenever a friend leaves his workplace.

IV. System Implementation

The described concepts have been implemented in a prototype implementation and tested with the RESTful Ploggs and the RESTful Sun SPOTs [2]. The actual implementation is composed of three distinct parts as illustrated on Figure 1. The core of SAC, a number of Publisher Components and applications built on top of SAC. The core of the system is SAC which implements the authentication and access control. SAC can be extended through *Social Network Drivers* which allow users to write custom drivers for additional social networks accessible through Web APIs. We have implemented two drivers, one for Facebook which uses the Facebook Connect API[8], and one for Twitter which uses the Twitter API.

Similarly SAC can be extended, through Publisher Components, to support various systems to which syndicated feeds of shared data can be redirected. We implemented an Atom Publisher Component which allows for publishing content coming from any shared resources to any Atom server implementing the ATOM Publishing Protocol, also known as ATOMPub[9]. We further implemented a Twitter Publisher Component which allows smart things to post data to Twitter accounts.

SAC is implemented on top of RESTlet[10], which is a comprehensive Java framework from implementing RESTful Web applications. SAC offers a comprehensive and RESTful Web API allowing new application to be built easily on top of SAC, by using the RESTful interface to: share, add, manage, syndicate, and interact with shared things. This makes SAC an integral part of the Web of Things since its API is accessible on the Web and can be used to build physical mashups. For example, a new Smart Gateway or smart thing can be added to SAC for sharing by sending a POST request to .../gateways alongside with the data of the new element (URL, name, description, etc.) as payload.

We also developed FAT: Friends and Things, which is an example of application that can be built upon the RESTful API of SAC. FAT is a Web application that allows users to share and use shared smart things in a user-friendly manner. FAT is a JavaScript Web application implemented using the Google Web Toolkit and provides Web-based user interface to the main functionality of SAC. It allows for users to login with several social networks at once, add new Smart Gateways and smart things (see Figure 2), share them with trusted connections or create feeds from several smart things.

V. Discussion

Bringing devices on the Web paves the way for a new breed of applications that seamlessly blend the physical world with existing Web applications. SAC and FAT are simple examples of the possibilities enabled by the Web-enablement of physical objects. Thanks to full integration of smart things to the Web, things can be shared based on the social structures encapsulated in social networks, by leveraging the fast developing Social APIs.

However, this approach also presents some challenges. First, applications built using these APIs are also strongly coupled with the social network platforms, and so are their users. For example, a Facebook user cannot extract his social structure to move it to another network. Additionally, social networks APIs offer different functionalities therefor it is difficult to group them under a single common denominator.

Several initiatives attempt to solve these problems. Among them, OpenSocial[11] defines a common API for social applications across multiple platforms. Initially started by Google, many social networks such as Orkut, Linkedin, Netlog, Yahoo!, Hi5, Myspace and many others have also joined and implemented the standard in their APIs. We first planned to support OpenSocial only in SAC, but while the initiative has a lot of potential, the current implementations are not entirely homogeneous yet and still under construction for most of them. Moreover, important actors such as Twitter and Facebook do not comply with OpenSocial as of today. We believe this is because of the strategic implications of such a standard. OpenSocial allows for users to take their profiles with them and move them across social networks. This goes against the "Walled gardens" [17] strategies of some social networks, a strategy that basically consist of preventing users to use their own data outside of the closed-walls of the social network. This situation forced us to define an extensible component model for which concrete social network connectors need to be implemented for each non-OpenSocial compliant network. Loosening access to user-generated data will be a significant but necessary step to move towards a truly Open Social Web.

This project also illuminated other important challenges to consider for making the Web of Things ready for opportunistic programming [7] and physical mashups. Applying a RESTful architectures to the Web of Things is an important step towards integrating smart things, however, further extensions are required. An important one is to support service description, which a key feature for usability and sharing. The crawling approach proposed here is limited as it requires an HTML page describing and linking the resources together, but without

[8] http://developers.facebook.com/connect.php
[9] http://tools.ietf.org/html/rfc5023
[10] http://restlet.org
[11] http://www.opensocial.org

enforcing a particular structure. This "fuzzy description" has worked well for search engines for years and is the way most RESTful APIs are still described (e.g. the del.icio.us REST-like API[12]). Nevertheless, crawling such non-structured documents might lead to irrelevant and non-compliant descriptions. Uses cases considering (automated) machine to machine communication (for example a sensor communicating with some actuators), would particularly be affected by this procedure and will require more standardized resource descriptors.

We suggest exploring the use of Microformats[13] to annotate human-oriented Web pages with key information encoded in a machine-readable format. Microformats have recently been used for describing RESTful services [18]. We are currently experimenting how to use a similar approach for describing resources of on smart things. In the case of sharing resources, this could greatly improve the user experience as it would help better describing what the resources are and what they can do.

VI. Conclusion and Future Work

The Web of Things needs sharing and security mechanisms. In this paper we have presented a system for sharing and controlling access to resources in a Web of Things. The core idea is to leverage existing online social structures rather than relying on closed databases of credentials. Thus, SAC builds upon fast growing social networks such as Facebook, Twitter and OpenSocial to allow users to share physical objects with actual friends, relatives or colleagues.

SAC also provides a programmable basis upon which composite Web applications can build. Thanks to the RESTful API of SAC, physical mashups and other Web applications can use SAC's functionality to share and use shared smart things. To illustrate this we introduced the Friends and Things Web application which directly builds upon SAC's API and tested it with two different types of smart things.

The development of SAC has shed light upon challenges for building an ecosystem of shareable things, for example the need for a comprehensive description framework for smart things. Future work will include evaluation of SAC and the FAT application in large-scale scenarios with actual users and a number of shareable smart things. This is especially important since SAC acts as a proxy between things and clients, thus needs to be scalable to support sufficient concurrent connections from many real users. The security aspects of the system also need further evaluation as some weaknesses in the APIs it is based on might open the doors to attackers.

Nevertheless, SAC illustrated the benefits of intergrating smart things to the Web. It shows how easily original applications bridging the fantastic developments of today's Web and tomorrow's Web of Things can be created. It is one step towards the vision of a composable and mashable ecosystem where smart things can be shared, used and re-used in a serendipitous manner.

[12] `http://delicious.com/help/api`
[13] `http://microformats.org`

References

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

[2] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices," in *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, Madrid, Spain, April 2009. [Online]. Available: http://www.vs.inf.ethz.ch/publ/papers/dguinard_09_WOTMashups.pdf

[3] E. Wilde, "Putting things to REST," School of Information, UC Berkeley, Tech. Rep. UCB iSchool Report 2007-015, November 2007.

[4] V. Stirbu, "Towards a RESTful Plug and Play Experience in the Web of Things," in *IEEE International Conference on Semantic Computing*, Aug. 2008, pp. 512–517.

[5] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[6] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding mashup development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[7] B. Hartmann, S. Doorley, and S. Klemmer, "Hacking, mashing, gluing: Understanding opportunistic design," *Pervasive Computing, IEEE*, vol. 7, no. 3, pp. 46–54, 2008.

[8] A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: an infrastructure for shared sensing," *IEEE Multimedia*, vol. 14, no. 4, pp. 8–13, 2007.

[9] T. Schmid and M. B. Srivastava, "Exploiting social networks for sensor data sharing with SenseShare," Oct. 2007. [Online]. Available: http://repositories.cdlib.org/cens/Posters/340

[10] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards Physical Mashups in the Web of Things," in *Proc. of the 6th International Conference on Networked Sensing Systems (INSS 2009)*, 2009.

[11] J. Hui and D. Culler, "Extending IP to Low-Power, wireless personal area networks," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 37–45, 2008.

[12] S. Duquennoy, G. Grimaud, and J. Vandewalle, "The web of things: interconnecting devices with high usability and performance," in *Proc. of the 6th IEEE International Conference on Embedded Software and Systems (ICESS'09)*, HangZhou, Zhejiang, China, May 2009.

[13] D. Yazar and A. Dunkels, "Efficient application integration in IP-based sensor networks," in *Proc. of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Berkeley, CA, USA, Nov. 2009. [Online]. Available: http://www.sics.se/adam/yazar09efficient.pdf

[14] V. Trifa, S. Wieland, D. Guinard, and T. M. Bohnert, "Design and implementation of a gateway for web-based interaction and management of embedded devices," in *Proc. of the 2nd International Workshop on Sensor Network Engineering (IWSNE 09)*, Marina del Rey, CA, USA, Jun. 2009.

[15] L. Richardson and S. Ruby, *RESTful Web Services*, illustrated edition ed. O'Reilly Media, Inc., May 2007.

[16] D. Hinchcliffe, "Open APIs reach new high water mark as the web evolves." [Online]. Available: http://blogs.zdnet.com/Hinchcliffe/?p=215

[17] T. Berners-Lee, "Twenty years: Looking forward, looking back," Madrid, Spain. [Online]. Available: http://www2009.eprints.org/212/1/index.html

[18] J. Kopeck, K. Gomadam, and T. Vitvar, "hRESTS: an HTML microformat for describing RESTful web services," in *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE Computer Society, 2008, pp. 619–625.